U.PORTO

MESTRADO
MULTIMÉDIA - ESPECIALIZAÇÃO EM MUSICA INTERATIVA E DESIGN DE SOM

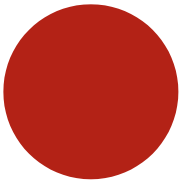
Aural exploration of post-tonal music theory: an automatic musical variations generator in MAX

Alonso Torres-Matarrita

M
2018

FACULDADES PARTICIPANTES:

FACULDADE DE ENGENHARIA
FACULDADE DE BELAS ARTES
FACULDADE DE CIÊNCIAS
FACULDADE DE ECONOMIA
FACULDADE DE LETRAS



FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Aural exploration of post-tonal music theory: an automatic musical variations generator in MAX

Alonso Torres-Matarrita

Mestrado em Multimédia

Supervisor: Rui Luis Nogueira Penha

Second Supervisor: Gilberto Bernardes de Almeida

July 17, 2018

Aural exploration of post-tonal music theory: an automatic musical variations generator in MAX

Alonso Torres-Matarrita

Mestrado em Multimédia

Approved in oral examination by the committee:

Chair: André Monteiro de Oliveira Restivo
External Examiner: Filipe Cunha Monteiro Lopes

July 17, 2018

Abstract

Some tendencies in post-tonal music are regarded as very “cerebral” or “cryptic”, and are often misunderstood, when not rejected, by the uninitiated public and even by some professional musicians. As an effort to overcome my resistance to this languages, I designed an interactive application in MAX that allow to improvise using Allen Forte’s set theory. So, rather than being forced to do a bunch of arithmetic calculations by heart before play a single note, musicians can hear instantly the aural qualities of determined pitch class sets by playing a simple chord in a MIDI keyboard to generate variations of a previously played musical gesture. An iterative design process was carried on, and the results, conclusions and reflections during the different stages of the work showed the relevance of the music theory applied and its relation to constraint satisfaction techniques in music composing processes.

Keywords: Musical variations, post-tonal music, computer assisted composition, contour theory, interactivity.

Resumo

Algumas tendências na música pós-tonal são consideradas muito "cerebrais" ou "crípticas" e, usualmente, acabam por ser incompreendidas ou rejeitadas pelo público não experiente, até mesmo, alguns músicos profissionais. Com a intenção de superar a resistência a estas linguagens musicais, criei uma aplicação interativa no MAX que permite improvisar utilizando a teoria de conjuntos de Allen Forte. Desta forma, em vez de serem forçados a fazer muitos cálculos aritméticos de cor, antes de poder tocar uma ideia musical atonal, os músicos poderão, instantaneamente, ouvir as qualidades sonoras de um pitch class set apenas tocando um acorde simples no teclado MIDI, para gerar variações instantâneas de um gesto musical tocado anteriormente. Foi realizado um processo de design iterativo onde os resultados, conclusões e reflexões das diferentes etapas do trabalho, mostraram a relevância da teoria musical que foi utilizada, bem como, a sua relação com técnicas de satisfação de constrangimentos nos processos de composição musical.

Palavras-chave: Variações musicais, música pós-tonal, composição assistida por computador, teoria de contornos, interatividade, MAX.

Acknowledgements

To Rui Penha, not only for his wise guidance during the writing and development of this work, but also for his inspiring classes, his human qualities, and all his generous help and support even before I took the first plane to Portugal. There are thousands of Ruis in Portugal, but none like you, thank you from the bottom of my heart.

To Gilberto Bernardes, for his authenticity, commitment and valuable contributions to this work, my big and honest thankfulness.

To my loving wife and partner Lilly, for all her patience, love and support. My life with you has been more wonderful than I can possibly ever imagine. What's our next adventure?

To Margarida, a true friend. Thank you for being you all the time, showing me the better side of Portuguese culture. I love you my friend.

To my colleagues in the Master of Multimedia, specially to Valter, João, Guida and "papi" Pedro. São porreiros demais.

To the teachers from the Interactive Music and Sound Design specialization. You're certainly the best guys in the MM.

To the people of SMC group for their support and feedback.

To OACIE at University of Costa Rica, without their support this would never been possible.

To my family in Costa Rica, I love you and miss you every day.

Alonso Torres

Contents

List of Figures

List of Tables

Abbreviations

AIA	Adaptive Interval Approximation
CAC	Computer Assisted Composition
CAAC	Computer Assisted Algorithmic Composition
CC	Contour Class
CIGW	Contour Interval Guided Walk
CIS	Contour Interval Succession
CSP	Constraint Satisfaction Problems
ICMS	Interactive Computer Music System
IRCAM	Institut de Recherche et Coordination Acoustique/Musique
LISP	List Processor (progamming language)
llll	LISP-like linked list
MIDI	Music Instrument Digital Interface
MIR	Music Information Retrieval
NCA	Normalized Contour Approximation
OM	Open Music
PC	Personal Computer
PCset	Pitch Class Set

Chapter 1

Introduction

Some tendencies of post-tonal music are regarded as very "cerebral" or "cryptic", and are often misunderstood, when not rejected, by the uninitiated public and even some professional musicians. This is still a controversial subject in academic music circles, but resistance to this music could be partially explained because of the difficulty to perceive relationships among the sound materials in, as composer Trevor Wishart pointed "the unidirectional experiential time"[Wishart, 1996]. It has been stated by composer Steve Reich (b.1936) that the processes of music construction in the works of John Cage or atonal serialists like Arnold Schoemberg or Anton Webern can not actually be heard in their music[Reich, 2004], and as Wishart stated: "The experience that the listener has *is* the music and the composer's methodology, no matter how rational it may be, is a heuristic device realizing the source of that experience in sound."[Wishart, 1996]. Though, the aesthetic leap produced in the twentieth century by breaking free from tonal restrictions seemed an inevitable breakpoint in the evolution of art music history[Kostka, 2006].

I myself was very reluctant to explore post-tonal languages in my compositions, but as time passed, I started to feel a necessity of extending the boundaries of my music composing practices into new means of expression. While studying procedural generation of music I realized the relevance of twentieth century music theory, and also its intimate relationship with many algorithmic procedures I was studying. This work is an effort to overcome my own resistance to this languages, and a tool for other musicians to smoothen the exploration of post-tonal music tendencies.

1.1 My motivation

As a composer, hands-on improvisation of music materials is my favorite approach to develop music ideas and to find aurally satisfying solutions to different composition problems. During my academic music studies, while addressing post-tonal music language, I realized that improvising music fluently in consonance with the theoretical principles of post-tonal theory could be very challenging. As many arithmetic calculations are required to do a theoretically coherent pitch selection, the process of writing in this style appeared to me as less fluent, spontaneous or organic than that of writing in other music styles I was more familiar with.

During the classes of procedural music generation in the master of Multimedia at University of Porto, I realized that pitch class set theory (the standard for analyzing post-tonal twentieth-century concert music) could be smoothly implemented in computer processes. When I tried to implement pitch class set theory procedures with computer algorithms I started to notice how fast this implementations allowed me to hear the particular aural qualities of post-tonal sonorities. That motivated me to experiment with chords and intervalic combinations that I don't usually chose for improvising. I dared to try sonorities I was not familiar with, in a process seasoned by many serendipitous moments.

From the desire of integrating improvisation, post-tonal music and automatic music generation came the motivation for this work. When learning how to improvise, musicians are usually taught to recreate, in their instrument, a musical idea that was first devised in their imagination. This process that goes from an aural image to its actual performance in a musical instrument requires an automatized articulation of musical mechanics that can take years of training. In my musical practice experience, when trying to improvise in post-tonal languages, pitch selection is the most challenging feature of the task, but usually I had a very clear idea of the rhythm and the general contour of the music I wanted to play. Hence, I thougth it would be useful to create a system that generated a version of a musical idea that was played in a MIDI keyboard only changing it's pitch content. It should leave the rhythm exactly as it was played and the general contour of the gesture should be reinterpreted according to defined pitch combinations.

1.2 Current alternatives

A system like that I described in the beginning of this introduction is somewhat related to two well-known technologies that are present in many commercial music keyboards: arpeggiators and automatic accompaniment systems. Nevertheless, none of this technologies was able to properly accomplish the objectives of my work. Arpeggiators generate looped music patterns from a user input, generally a sustained chord or note. The pattern is looped while the user holds the chord and stops when all keys are released. In modern keyboards, such as Korg's Kross Music Workstation[1] users can create custom arpeggiator patterns by filling a rhythmic grid space like that of figure 1.1. Nevertheless, the gestures are edited offline, the rhythmic grid impose rhythm restrictions and the way contours are reinterpreted make a MIDI keyboard arpeggiator too limited to use it as a solution to the problem. In regard to automatic accompaniment, even though many keyboards like the PRS models by Yamaha allow custom recording of patterns that are reinterpreted according to a user input (cfr. [Yamaha, 2013]), such systems are constrained to the tonal language, and usually the custom patterns to create variations are assumed as being played inside a C major macro-harmony to be reinterpreted tonally. Since my interest resides in post-tonal music, an automatic accompaniment system like this is neither suitable for the objectives of my work. That made clear that there were space for a contribution in this matter.

[1] http://www.korg.com/us/products/synthesizers/kross2/

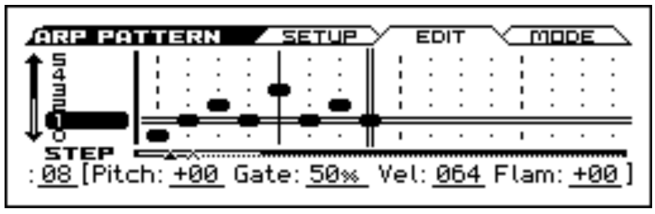


Figure 1.1: Korg's Kross arpeggiator design grid. [Korg, 2013]

1.3 Research objective and questions

Since I did not find any system that made a similar work, this study is an exploratory one, having the following qualitative objective:

> **To explore the development of a system that replicates, in real time, the interval contour of a musical gesture played in a MIDI keyboard, with user defined pitch class set restrictions.**

This objective can be achieved by answering the following research questions:

- What music information retrieval (MIR) procedures can be used to capture and store contour information from a performance in a MIDI keyboard?
- What procedures can be used to reinterpret a musical gesture contour with user defined pitch class set restrictions in real time?
- What relations can I establish between different constraint satisfaction procedures and the aesthetic/aural results of the gesture transformations produced?

1.4 Research Methodology

Given the exploratory character of this work and that my professional background is in art and music composition, I followed an iterative art-practice research methodology. This qualitative methodology is included in what Archer named action research, that is when a systematic enquiry is conducted through the medium of practitioner activity [Archer, 1995]. López-Cano and San Cristóbal go further describing specifically art-based research in music stating that this research activity produces knowledge and tools for the development of the music practice, which includes music creation software and multimedia resources [López-Cano and Cristóbal, 2013]. This dissertation intended to generate knowledge about the use and design of a computer tool that facilitates an aurally significant exploration of gesture transformations applying post-tonal music theory.

Art-based research is systematically conducted and explicit in its methods. Like in other qualitative methodologies, a researcher can explicitly take action in and on the real world in order to devise or test or shed light upon something. I was totally involved in the creation and testing of

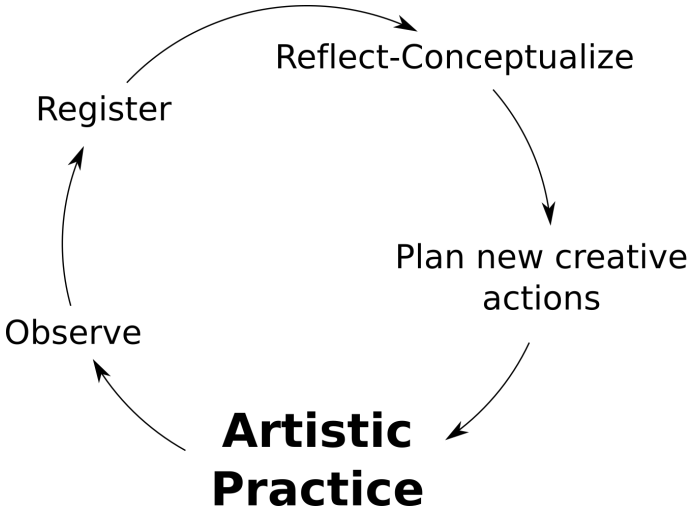


Figure 1.2: Interaction and feedback between creative practice and reflection. [López-Cano and Cristóbal, 2013]

the tool, and all results from this exploration were systematically documented in a detailed log. All the steps taken and my methods of creation and my reflection about the results intended to be transparent and explicit. López Cano and San Cristóbal synthesize this process in a loop illustrated in fig 1.2.

In this work, a systematic register of reflections and actions taken is the center of the methodology. Every work session is detailedly registered, having three sections that correspond to register, reflection and planning. Register section is a record, includes: modifications in the code, results obtained, difficulties and advances. Reflection section is an analysis of what was done: questions are generated, concepts are traced, it is a general evaluation of the implementations made and a opportunity to formalize emerging knowledge. Finally, the planning sets new paths and objectives for the upcoming sessions, helping to keep continuity and focus through all the work.

1.5 The structure of this document

The following chapters are organized as follows: chapter two deals with the aspects of Post-Tonal theory that are the foundation of this work. Chapter three includes the antecedents and state of the art in computer assisted composition, algorithmic composition and interactive music systems. Chapter four is an organized version of my working log, and summarizes the iterative process of design. Finally, Chapter five presents the results and conclusions, and traces some possible lines of future work.

Chapter 2

Post-Tonal Music Theory

This chapter summarizes in two sections the elements of Post-Tonal theory that are implemented in this work. First, a link between tonal harmony and post-tonal pitch organization is established, to later introduce the set theory of Allen Forte as elaborated by Joseph Straus. The second part deals with theory of contour, a tool for understanding and analyzing the pitch profile of music gestures in time. Finally, a third section summarizes some implementations of set theory that are relevant as antecedents for this work.

2.1 Antecedents and pitch organization in Post-Tonal music.

For centuries, the effects of pitch combination have attracted the attention of musicians around the world, to the point that the study of different practices and techniques in regard to simultaneously sounding pitches (harmony) became a fundamental subject in the formal study of academic music. In the context of european art music, during the Baroque, Classical and Romantic periods, the so called "tonal system" was the primary organizing force of music composition, and it is still present in commercial music, jazz, rock and many other musical styles[Kostka, 2006].

During the period known as "of the common practice" (approximately between 1700 and 1900)[Gauldin, 2009] though tonality prevailed in occidental art music, the emphasis on dissonance, chromaticism and tonal ambiguity increased gradually and decidedly [Kostka, 2006]. By around 1900 the tonal system was so strained by chromaticism and the desire of originality that further development of the tonal system seemed impossible. As part of this apparently inevitable process, some representative occidental art music composers wrote music that avoided the conventional melodic, harmonic an rhythmic patterns that helped to establish a tonal center in the common practice, moreover, this music had a tendency in the exhaustive use of the chromatic scale [Kostka, 2006]. The so-called post-tonal music required new analysis tools and to this respect the theory stated by Allen Forte and later updated by Kostka and Straus became standard.

Integer name	Pitch-class content
0	C, B♯, D𝄫
1	C♯ , D♭
2	D, Cx, E𝄫
3	D♯, E♭
4	E, F♭, Dx
5	F, E♯, G𝄫
6	F♯, G♭
7	G, Fx, A𝄫
8	G♯, A♭
9	A, Gx, B𝄫
10	A♯, B♭
11	B, C♭, Ax

Table 2.1: Integer notation for pitch-classes.([Straus, 2005] p.5)

Due the essential importance of this tools for this work, the more relevant aspects post-tonal theory for this work are explained below.

2.1.1 Pitch classes

Post-tonal music is organized around the selection of pitches or intervals in a fashion that not responds to the well-known major or minor scales and tonality rules and tendencies. For that sake, some particularities in regard of enharmonic equivalence are take in account when it comes to pitch representation in post-tonal theory. Departing from the equal temperament[1] a D♭ and a C♯ are considered equivalent, since their pitches in the same octave will be the same.

A further abstraction of this idea is to consider all notes of the same name as belonging to the same pitch class. Since all musical notes with the same name and their respective enharmonics belong to the same pitch class, there are only 12 pitch-classes, corresponding to each note of the chromatic scale. This is also referred as octave equivalence [Straus, 2005].

In post-tonal theory, pitch classes are better represented by integers rather than note names, this simplifies further operations and musical analysis in post-tonal music. Arbitrarily, the note "C" corresponds to 0 and from there the other integers are assigned to the rest of pitch-classes (idem, p.5).(see Table 2.1)

2.1.2 Interval classes.

General speaking, an interval is the pitch difference (referred as "distance") between two notes [Gauldin, 2009]. While in tonal practice the interval classification is defined both by the amount of semitones that separate those notes and by the diatonic scale steps involved in that relation, the post-tonal approach considers only the semitone contents, two intervals that sound the same are considered the same, no matter how they are written. To classify intervals, one can consider

[1] Equal temperament is used in the western art music approximately since around 1800 and means that all semitones have "the same size"[Gauldin, 2009], in opposition to other temperaments where semitones were asymmetric.

Diationic Simple Interval Reference	Interval class
Unison, octave	0
Minor second, major seventh	1
Major second, minor seventh	2
Minor third, major sixth	3
Major third, minor sixth	4
Perfect fourth, perfect fifth	5
Augmented fourth, diminished fifth	6

Table 2.2: Integer notation for interval classes.([Kostka, 2006] p.186)

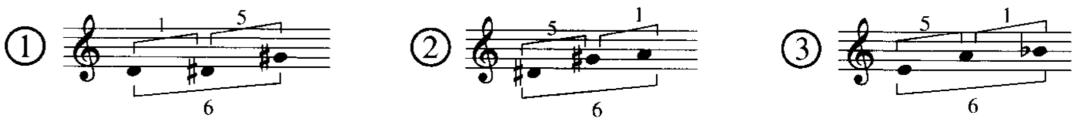
both the total distance of semitones between the two notes and also the direction of that skip from one note to the last one. In a further level of simplification, the direction will be put aside and the distance be will synthesized to the range of an octave; this is because all compound intervals can be expressed as simple intervals with one or more octaves added. In the interval class categorization intervals that share an inversion relationship are considered equivalent. This leave only 7 interval classes that are also expressed with integer notation: 0, 1, 2, 3, 4, 5, 6. These numbers correspond to the semitones that separate the smaller member of the inversion pairs as were referred (see Table 2.2).

2.1.3 Interval class and interval class vector

Each of these interval classes has a particular sonic quality, a character. The psychoacoustic effect of hearing a minor second or a perfect fifth is very different and can be explained by the amount of relation or coincidence that is perceived between the partials of two pitched sounds [Parncutt, 1989]. While atonal music arrives as the culmination of a growing freedom in the use of dissonance, the sound (understood as the human perception) of dissonance did not changed. From a perceptual point of view, the traditional classification of intervals into sharp dissonances, mild dissonances, rich consonances, and open consonances still remains valid since it remains audible [Belkin, 2018]. That being said, the particular sound character of a given set of pitches sounding together is given by the interval present in each possible two pitch-class combinations. In atonal music theory, an inventory of all the resulting interval classes between all pitches of one sonority is called the interval class vector. The number of interval classes a sonority contains depends on the distinct pitch classes in combination, e.g. a sonority with three pitch classes has three interval class to count, but one with five will have ten interval classes. The interval class vector is usually represented in a scoreboard fashion, indicating in the appropriate column the number of occurrences of each of each interval classes (all two note possible combinations count) excluding the occurrences of interval class 0. All intervals contribute to the overall sound. (see Fig 2.1)

2.1.4 Pitch Class Sets

It has been recognized that atonal music often achieve certain degree of unity by using a new particular kind of motive or elemental music cell. This is what is called in atonal theory a “pitch-class



interval class	1	2	3	4	5	6
no. of occurrences	1	0	0	0	1	1

Figure 2.1: Three different sonorities with the same interval class vector. ([Straus, 2005], p. 13)

set" ("PCset"), and can be compared to a building block in an atonal composition. A pitch-class set is an abstract collection of two to eleven pitch classes where characteristics like timbre, register and rhythm are boiled away. In an atonal composition, one pitch-class set could appear melodically, harmonically or as a combination of both, in any order or register or also transposed. The analysis of atonal music usually includes the process of identifying and labeling these important pitch class sets in a process that involves segmentation ([Straus, 2005], [Kostka, 2006]).

2.2 Elements of contour theory

Contour in post-tonal music is a wide area of research that has attracted a lot of attention in academic and compositional circles. In the eighties, Marvin stated that pitch class set theory remains too limited to model a listener's aural perception of sounding music [Marvin, 1988]. Friedman in turn, defended that perception of contour is more general than perception of pitch, interval and set classes[Friedman, 1985], since contour perception is only grounded in the ability of a listener to hear pitch as relatively higher equal or lower [Morris, 1993]. Several studies determined that listeners are more likely to identify contour variations that pitch alterations when comparing a melody to a modified version. All this support the main assumption of this work that contour and rhythm are the most recognizable aspects of a musical idea, and modifying only pitch content of a musical gesture is a convenient way of assessing the aural qualities of a particular pitch class set keeping a strong link in different transformations of the same musical idea.

Generally speaking, contour refers to "An outline representing or bounding the shape or form of something, [or a] way in which something varies, especially the pitch of music or the pattern of tones in an utterance."[Oxford, 2018], describing usually how pitch or other music property changes on time. Most references cite [Schoemberg, 1967] and [Toch, 1948] as early considerations of music contour in the twentieth century. The works of [Friedman, 1985], [Marvin, 1988], and [Morris, 1993] made fundamental advances in regard to contour theory that were recently developed and extended by [Bor, 2009], and [Schultz, 2016]. Contour analysis can be applied not only to pitch, but to density, duration, and not necessarily considered always against time. Nevertheless, since this work is focused on the pitch content of music gestures, contour will be considered as the shape described by pitch changes in time, represented as "an ordering of relative

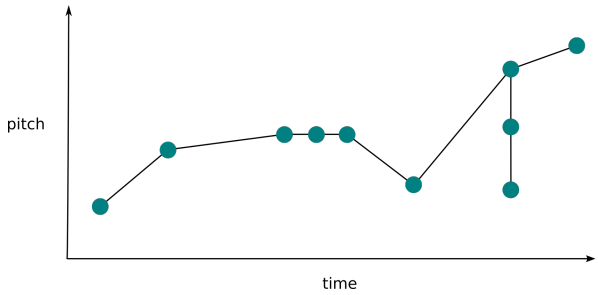


Figure 2.2: Representation of a music contour in a two-dimensional space.

pitch height (y-axis on a Cartesian coordinate system) in relation to sequential temporal ordering (x-axis on a Cartesian coordinate system)"[Bor, 2009] p.1 like in figure 2.2.

Two well established music contour analysis and description tools are used in the algorithms developed in this work. They were proposed originally in [Friedman, 1985], and are still at the foundations of modern contour theory developments (cfr. [Bor, 2009] [Schultz, 2016]). Next paragraphs deal with this concepts.

2.2.1 Contour adjacency series (CAS):

It is a simple yet effective tool to describe the general direction of pitch changes in a melodic profile. Given a series of pitches, one can describe the pitch variation between one pitch and another through ordered pitch intervals [Straus, 2005], which indicate both the direction (using + or - signs) and the magnitude in semitones of the variation. CAS deprives the indications from the number of semitones concerning only with direction, thus a CAS of a given melody is a string of +'s and -'s, ignoring adjacent repeated pitches, since there is no variation. (see figure 2.3). This is a general representation of pitch movement, and there is no way of judging the contour relation between non consecutive pitches [Friedman, 1985].

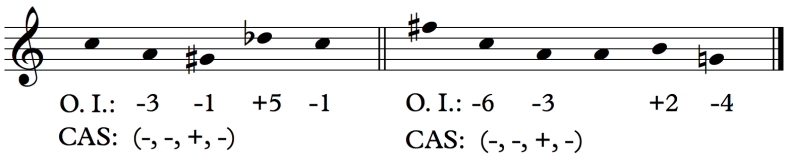


Figure 2.3: Two melodic contours with the same CAS.

2.2.2 Contour Class (CC):

It is a more inclusive and specific account of the contour pitches in a music idea. It reveals also the relation among all contour pitches and the occurrence of pitch repetition. The use of contour class requires a complete acknowledgment of the musical unit analyzed, since it reveals the position of pitches in relation to one or another. A CC is a string of numbers. For each

note, a number from 0 to $N-1$ is assigned, where 0 is the lowest pitch in the contour and N the number of different pitches in the musical unit [Friedman, 1985]. Contour class indicates clearly the internal relations between all notes in the profile, though interval distances and pitch classes are disregarded. Figure 2.4 shows the CC of the previous examples.

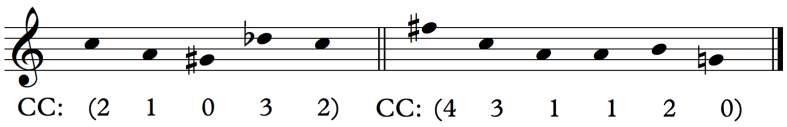


Figure 2.4: CC of two melodic profiles.

2.2.3 Contour Intervals and Contour Interval Succession:

A contour interval measures the distance between two contour integers in a CC. A Contour Interval Succession (CIS) is the list of successive contour intervals in a particular CC. For instance, a CC that is (1, 4, 5, 2, 3, 0) will have a CIS of (+3, +1, -3, +1, -3). [Bor, 2009]

2.2.4 Normalized contour:

In a recent article [Schultz, 2016], a normalization of CC representations has been proposed to override some ambiguities and misunderstandings the traditional CC number representation can induce to. Since CC integer notation accounts for a relative pitch ordering disregarding interval distance, comparing the CC integers of two profiles with the same range but different amount of contour components, can induce to think that the profile with more elements goes somewhat higher than the other one. This is due to the fact that in the most diverse contour the highest pitch is represented by a bigger integer than in the other one. E.g. If contour segments (2 3 0 1) and (5 7 6 2 1 3 0 0 4) had the same highest and lowest pitches, the second number string gives an impression of expansion, which is not necessarily true. To override this, Schultz propose a normalized representation where 0 represents the lowest pitch of the contour segment, and all digits from CC are divided by the anterior $N-1$, that is, the cardinality of the contour pitches less one. That way, 1 will always correspond to the highest pitch in the contour. E.g. a CC with five contour pitches (0 1 2 3 4) will be represented as (0 0.25 0.5 0.75 1). This kind of normalization allows to have always the same digit to represent the highest and the lowest pitch, no matter the cardinality of contour pitches in a CC.

2.3 Assistive implementations of PCset theory

As was stated previously, post-tonal music theory is often a frightening subject in academic music studies. In this concern, some effort has been made in order to demystify post-tonal theory, or at least facilitate its comprehension. In an article by Stanley Kleppinger, professor at University of

Nebraska–Lincoln, an experience with post-tonal theory is referred which appeared very familiar to me:

> During an office hour many years ago, an above-average student in my undergraduate theory course asked me what topics remained for us to cover in that semester. When I mentioned pitch- class set theory, she asked with great trepidation, "Is that the atonal music with all the math in it?" [...] As I learned, this student was attempting to articulate two fears: that the music we were about to study would sound like cacophonous nonsense, and that to understand it she would need to be a mathematician. [Kleppinger, 2010] p.131

Professor Kleppinger describes later in his article a methodology that he claims lead students to discover by themselves the necessity of post-tonal theory by analyzing a selected work by Anton Webern using common practice theory until they finally give-up and the necessity of new analysis tools emerges.

Besides that, two works were found that have a more technological approach, the first one is the "Atonal Assistant" [2], a relatively old website that performs pitch class set theory operations. The user enters a PCset in the form or integers or musical notes and the system performs calculations e.g. retrieve the prime form, the interval class vector and the Forte name (see figure 2.5). Though very useful, this is merely a statistical tool, and does not offer any playback or aural feedback.

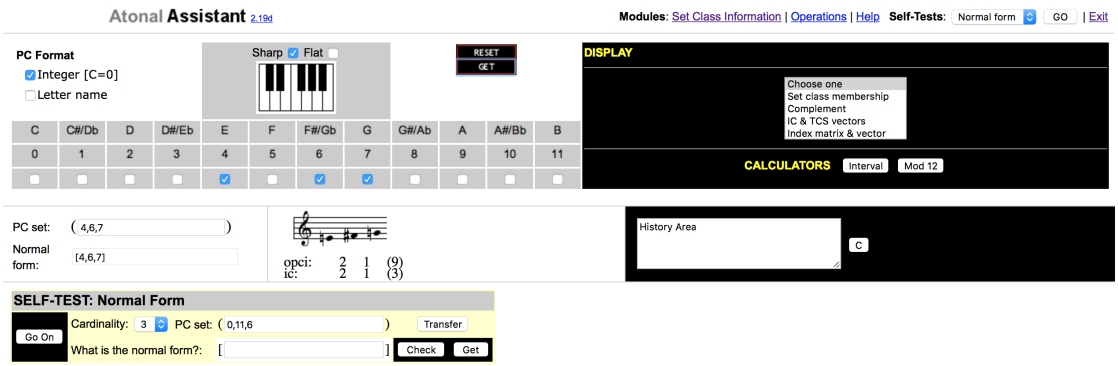


Figure 2.5: User interface from the Atonal Assistant website.

There also exists the *music21* library for python [Cuthbert and Ariza, 2010]. This library is a powerful collection of python objects that is essentially musicology oriented. Includes tools to display music notation, detailed search strategies from a pre-built corpus of music in symbolic representation and the capacity of performing several statistical operations. Some generalities of post-tonal theory are included, such as objects that show information from a PCset and generate twelve-tone row matrices. It seemed to be capable of generating variations in the way I pretend to do in this work, but it is not precisely an aural exploration tool. Among its drawbacks for using it were that playback works off-line. In order to play a music representation, output is obtained

[2] http://in.music.sc.edu/fs/bain/software/aa-v2.20d/default.htm.

as a XML file that should be imported by another software like Sibelius or Finale, so no real interactivity is allowed.

None of this works refer to a system that allows to generate music variations in real-time applying post-tonal theory, nor a technology assisted aural exploration of atonal sonorities. This confirmed the necessity of creating, from scratch, a system that addresses the objectives of this exploratory work.

2.4 Summary

When breaking with tonal harmonic language, art music composers opened to more free pitch combination procedures. New analytical tools were developed for this music, and set and contour theories proposed by Allen Forte and Michael Friedman, and extended later by authors such as Kostka, Straus, Marvin, Bor and Shultz reflect the rational and formalized design behind much of this music in a way that makes its computer implementation free-flowing.

Previous work found around assistive tools for post-tonal theory exploration lacks either of aural, creative or interactive approaches. That confirmed the necessity of creating a system from scratch employing technological resources whose context and historic development are summarized in the following chapter.

Chapter 3

Automatic music generation

This chapter is devoted to different aspects of automatic music generation. It overviews the development of this artistic practice since its beginnings to the present, including the current tendency of including more interactivity and real-time behavior in computer assited composition systems.

3.1 Computer aided algorithmic composition

3.1.1 Antecedents

Composing algorithmically, that is, writing music by means of formalized methods, is not a recent phenomenon in music nor an application of computer science to music composition (cfr. Roads et al. in [Guedes, 2008] p. 1 - 2). For instance, around AD 1000, in his work "Micrologus" Guido of Arezzo proposed a system to automatically assign pitches to a text according to the vowels present in it. ([Nierhaus, 2009] p. 21). Also, discrete symbolic representation of music parameters and formalized rules for music composition precede the invention of digital computers by a few centuries. The five-line system and the rhythmic figures in traditional music notation are a symbolic representation of discrete sound properties, and many of the musical knowledge in regard harmony and counterpoint has been formalized as a set of rules for pedagogical purposes.

Since discrete parameters and rule-base programming are things that computers do very well, formalized musical creation procedures can be programmed into computers to generate music automatically. The work of music composers and computer scientists around this possibilities is referred as computer assisted composition (CAC)[Assayag, 1998], algorithmic composition [Nierhaus, 2009], or computer aided algorithmic composition(CAAC)[Ariza, 2005]. The latter is the more specific and unequivocal term, since an example of mere computer assistance may be a music engraving software, which has nothing to do with the computer decision making in the composition. Also, talking only about "algorithmic composition" not necessarily implies the use of a computer (cfr. [Ariza, 2005]). CAAC then will be the term I adopt in this work for referring

to those processes of music composition where formalized methods are executed by a digital computer to generate music in a symbolic level[1], or as stated by Ariza "A CAAC system is software that facilitates the generation of new music by means other than the manipulation of a direct music representation" [Ariza, 2005, p.1].

In automatic generation of music, either with digital computers or not, stochastic and chance procedures have played a fundamental role. In this matter, *Illiac Suite* is an important historic reference. This work became famous as the first completely computer-generated composition on a symbolic level, and was developed by Lejaren Hiller and Leonard Isaacson from 1955 to 1956 on the ILLIAC computer at the University of Illinois [Nierhaus, 2009, p.64]. The work is a four movement string quartet. In its first three movements, sets of random generated numbers were treated as music properties and confronted to a set of musical rules expressed to select that sequences that better fit the corpus of rules, which were derived from the historic practice of counterpoint, harmony and twelve-tone technique. In the fourth and last part (each one of them was entitled as "experiments") Markov chains are employed to create sequences of musical events where the selection of a new one depends on the preceding events (Hiller et al. in [Guedes, 2008, p. 3-4]).

Stochastic procedures have influenced and sustained much of the development of automatic music generation. For instance, Markov models are used generate algorithms of corpus analysis imitation, or to desing probabilistic models *ex-professo* like those used by Iannis Xenakis in *Analogique A*. In that work, Xenakis established transition matrices to sequentially select predefined segments of music to generate a musical structure [Nierhaus, 2009]. Of particular interest are a series of pieces that he produced using the IBM 7090 computer in 1962 in France. Xenakis designed algorithms that formalize several aspects of the composition (e.g. general structure, note density, orquestration) and generated several pieces by manipulating input data. These pieces are: ST/48-1 for orchestra, ST/10-1 for ensemble, Amorsima-Morsima (ST/ 10-2) for ensemble (complement of ST/10-1); ST/4-1 for string quartet; Morsima-Amorsima for mixed quartet and Atrées (ST/10-3). (Harley in [Guedes, 2008, p. 5])

As a conclusion from that work, Xenakis devised the concept of composer-pilot who, by taking advantage of computer's calculation speed:

> ...is able to devote himself to the general problems that the new musical form poses and to explore the nooks and crannies of this form while modifying the values of the input data. For example, he may test all instrumental combinations from soloists to chamber orchestras, to large orchestras. With the aid of electronic computers the composer becomes a sort of pilot: he presses the buttons, introduces coordinates, and supervises the controls of a cosmic vessel sailing in the space of sound, across sonic constellations and galaxies that he could formerly glimpse only as a distant dream. Now he can explore them at his ease, seated in an armchair ([Xenakis, 1992] p. 144).

[1] By symbolic level it is understood that the output of the system should be interpreted as note values either by real musicians or by other music software.

This idea of using the computers to help the composer with operations required to materialize musical ideas is at the heart of further advances in what was called computer assisted composition. Beyond generating music automatically, this developments had the fundamental idea of enabling composers to take advantage of the computer capacities to enhance the possibilities of their music thinking. Gérard Assayag, currently the head of IRCAM's Music Representation Team clarifies this distinction when talking about *Musicomp* (Music Simulator Interpreter for Compositional Procedures), which he refers as "maybe the first software designed for assistance to the composition, in opposition to automatic composition"[Assayag, 1998]. This software was written by Robert Baker in the sixties with the expertise of Lejaren Hiller (who worked in *Illiac Suite*) and Assayag describes it as "a collection of tools for solving problems specific to musical composition, in the form of sub-routines in the FORTRAN language" [Assayag, 1998]. *Musicomp* is, then, the first from a family of specially devised software for CAAC to whom I will refer later.

During the seventies, CAAC was somewhat overshadowed by an interest in digital sound generation and synthesis. CAAC then experimented a revival in the eighties, attributable to the availability of personal computers, development of standards in inter-machine communications (e.g. the MIDI protocol), better graphic interfaces and progresses in programming languages [Assayag, 1998]. This certainly opened a window for many musicians to get more and more involved in the computational aspects of music creation assisted by computers. As Assayag states, the use of a programming language forces the musician to reflect on the process of formalization and avoids him to consider the computer as a black box which imposes its choices to him [Assayag, 1998]. Some pioneer interesting projects developed in IRCAM are worth to mention before we talk about the surviving classics:

- Formes (1985): addressed the integration between composition and sound synthesis using VLISP language and represented music object as software actors and control structures as message senders.

- Crime (1985): a general environment for CAAC (written in LISP) which was also capable of displaying results in traditional music notation.

- Carla (1989-1990): This software was rather a graphical interface for logical programming. It included a set of basic and extended types and a set of heuristics associated with each type graphical interface for formalizing the relations between types. This very elemental construction allowed the composer to generate their own system of semantics. [Assayag, 1998].

3.1.2 PatchWork, OpenMusic and PWGL

The creation by IRCAM of PatchWork and, later, OpenMusic consolidated the idea of visual programming tools for CAAC where the composer had the ability to control abstract aspects of the composition process. PatchWork is based in Common LISP, so any LISP function in the program can be translated into graphically operable boxes that can be visually connected through visual

lines to other boxes, defining the program's syntax. The PatchWork kernel consist of an extensive set of predefined boxes which are musically neutral since they don't make assumptions of the specific music material they will process. Like other graphic environments, PatchWork is event driven, so actions in the boxes are triggered by external events such as mouse clicking or key pressing. For 1999, the program included libraries to work with spectral music, sound synthesis, stochastic manipulation of music parameters, constraint based generation of music structures and rule programming. Some PatchWork boxes have a variety of editors to visualize particular musical data like notes, chords, non-metric chord sequences and voices in the form of a breakpoint function, musical notation, or editable cell arrays [Assayag et al., 1999].

Open Music (hereafter OM) for its part, sits on a higher of CLOS and is a visual interface for LISP and CLOS languages, and is also an object-oriented visual programming tool derived from the PatchWork environment. Both programs have different representations of musical processes like common notation, midi piano-roll, and sound signal. A remarkable feature of OM is the "maquette" concept, which enables high-level control of musical material over a time-line, where any objects can be disposed and by superposition and concatenation, the composer can create a time-based articulation of musical objects. Existing CommonLisp/CLOS code can easily be used in OM, and new code can be developed in a visual way. Another powerful feature of this software is the capacity of abstraction, where several objects can integrate a new object or abstraction facilitating multilevel organization of musical classes and subclasses. (OM web-site, [Assayag et al., 1999]).

Some contributors to the expansion of PatchWorks had developed PWGL, which is an OpenGL successor of PatchWork, as opposed to PatchWorks that was created using QuickDraw [Laurson and Kuuskankare, 2002]. Their creators describe them as a "visual programming language specialized in computer aided composition and sound synthesis, providing a direct access to its base languages, Common Lisp and CLOS" [Laurson and Kuuskankare, 2013]. PWGL is a powerful CAAC language with a strong integration between:

- Common LISP, CLOS functions and abstractions.
- Traditional and graphic musical notation.
- Sound Synthesis
- Constraint Satisfaction tools.

In the line of LISP developments for CAAC, it is very relevant to this work the *bach* library for MAX[2] created by Andrea Agostini and Daniele Ghisi. This collection of objects incorporate many features CAAC features previously present in PWGL or Open Music in an interactive environment, allowing composers to have immediate, real time and interactive behaviors with automatic music generation. The integration between CAAC system and interactive music systems is the matter of the following section of this review.

[2]MAX is an interactive music software created by Miller Puckette. Its features will be more broadly explained later, when discussing interactive music systems.

3.2 Interactive Computer Music Systems (ICMS)

3.2.1 Historical context

The development of faster digital computers favored the practice of implementing algorithms to create music by digital means in live performance in the past decades. Real-time response is the main characteristic of interactive computer music systems (ICMS), whose behavior changes in response to a musical input [Rowe, 1993]. Interactive computer music is regarded as a branch of algorithmic composition, differing in the fact that interactive systems are capable of changing their algorithm's behavior during its execution. These characteristics make them very attractive to use them in a performance context, where they can react to what is going on around them, including making human participation necessary for the music to work [Guedes, 2008], p. 12.

Among the pioneer works in interactive music composition, it is necessary to mention John Cage's work during the 30's and 40's and his interest in indetermination and improvisation which led a to a growing interest in interactive electronic systems. Max Mathews is another pioneer who together with F. Richard Moore developed the GROOVE system between 1968 and 1979 in Bell Laboratories. This system had a musical conductor program which allowed the user to control tempo, dynamics and balance of a set of computers that were to perform a defined piece. Finally, in 1969 Joel Chadabe in collaboration with Robert Moog produced the CEMS (Coordinated Electronic Music Studio) in New York. As part of their experiments, a big set of Moog synthesizers were controlled by a pseudo-aleatoric algorithm to automatize a full music piece. Chadabe used joysticks to control filters, modulators, amplifiers and several sequencers (Winkler, Dodge and Jerese and Chadabe in [Guedes, 2008]).

The work of Robert Rowe deserves a special mention, not only because he developed for the first time a taxonomy of interactive music systems, but also for his ICMS *Cyhper* (1987-1993). This software was capable of "listening" to a MIDI stream of notes and analyze it to extract musical information and react accordingly by generating musical responses, all that in real time. He divided his system in two parts: "listener" which analyzes, groups, and classifies input events as they arrive and "player" which generates music algorithmically after receiving messages from the listener. The system never played a pre-stored response. A particular characteristic of *Cypher* is that the user has a series of options to determine how the player will respond to the messages emitted by the listener[3], such as performing transformations, playing sequences or starting algorithmic procedures [Rowe, 1993]. His work on implementing music music theories computationally inspired many of the posterior developments in ICMS.

3.2.2 Taxonomy and Models for ICMS

The first taxonomy of ICMS appeared in Robert Rowe's Interactive Music Systems, and is based in three different dimensions "whose attributes help identify the musical motivations behind types of

[3] It is my speculation that this kind of mapping similar to a cyphering code gives the system its name.

input interpretation, and methods of response" [Rowe, 1993]. That dimensions are the following, as referred in [Drummond, 2009]:

- **Score driven vs. performance driven systems:** Score driven systems have embedded knowledge of the overall predefined compositional structure. They adapt to a player's performance as a virtual accompanist, tracking the performer's advance in the musical text and triggering musical events accordingly. This approach is considered as a reactive implementation of the tape-instrument duet. Performance driven systems, on the other hand, don't have a predefined musical structure to follow, and rather elaborate responses based only in the analysis of what the system hears.

- **Transformative, generative or sequenced methods:** This dimension describes how the system responds to what it hears. A transformative response implies an algorithmic processing and variation using techniques as filtering, inversion, retrogade, transposing, delay, re-synthesis, distortion and others. Generative methods use a set of rules to generate complete musical responses from very elemental materials. Stochastic selection, generative grammars and flocking algorithms are example of generative procedures that fit into this classification. Sequenced responses playback pre-constructed and stored music materials, usually with some transformations in response to the input.

- **Instrument vs. player paradigms:** This category classifies the system behavior either as an expanded instrument or as a player companion, tracing a line accordingly to the independence of the response. In an instrument paradigm, the system response to a given input is predictable, direct and controlled. The same performance gesture produces the same or almost the same output. If the system employs the player paradigm, that responses are more independent, behaving like a virtual improviser or partner, with some connection to the human player but also with some independence and autonomy.

Rowe's definition of interactive music systems refers to the technological means available back in 1993, when wrote his book Interactive Music Systems. At the time, the majority of music information was manipulated in the symbolic level, and restricted to the possibilities of the MIDI protocol. Today, sound processing capabilities of digital systems and developing of sensing devices allow the ICMS to be fed a wider range of musical and non-musical information[Drummond, 2009], such as bio-sensors, motion capture devices, video, and Internet retrieved information.

3.2.3 MAX/MSP and PureData

MAX is a software developed at IRCAM in Paris in the beginning of 1986, its main developer was Miller Puckette, who designed it to control the 4X, a powerful digital synthesizer created also by the IRCAM to manage sophisticated sound synthesis in real time. 4X was initially thought to respond to score-following applications developed to experiment with interactive automatons for music accompaniment. This synthesizer was difficult to program, so MAX was designed as a

MIDI controller running in a Macintosh computer. From there, MAX developed as a very versatile graphic programming language for real time control of MIDI devices. Under the direction of David Zicarelly, MAX was released as a commercial software by Opcode Systems in 1991 as a full featured program with an improved graphical interface and a series of multimedia capabilities like playing MIDI files. It has been adopted by musicians interested in interactive applications and real-time control [Winkler, 2001]. Today, MAX is commercialized by Cycling '74 and includes a digital sound processing unit, that performs real-time synthesis and algorithmic audio processing. Furthermore, it includes video processing libraries like Jitter, a multimedia graphics 2D and 3D tool [Cycling'74, 2018].

Puckette for his part, has also developed PureData, an open source software based on the same paradigm as MAX. It is also a visual program tool capable of controlling MIDI devices in real time, digital sound processing and multimedia capabilities. As it happened with MAX, PD also has been enriched by external objects written in C by members of their respective user's community and has gained enormous popularity.

3.2.4 Francois Pachet's continuator

A very relevant work in ICMS is Francois Pachet's Continuator [Pachet, 2002]. Pachet is currently the director of the Spotify Creator Technology Research Lab and former head of Sony Computer Science Laboratory in Paris. Continuator was intended as a bridge between interactive music and style imitation systems. Using a specially devised Markov model, the system is capable of learning any musical style and improvise or "continue" musical ideas played by a human performer. When using the system, a user first plays in a MIDI keyboard, while the system is analyzing what s/he plays. After detecting a pause, the Continuator generates an plays a variation of what the user played.

An efficient variable-order Markov model allows this system to override the classic limitations of Markov Models: their tendency to plagiarism and their limited generative power due to the absence of long-term information [Pachet, 2002]. Continuator's Markov sequences store pitch and rhythm information in optimized data trees that store indexed sub-sequences that summarize all possible element concatenations present in the models. By random walking through continuation indexes, in-style new sequences are generated [Pachet, 2002]. Results of the system at work interacting with live musicians[4], shown an accurate style imitation capacity.

3.2.5 The *bach* library for MAX

The *bach* library[Agostini and Ghisi, 2012] for MAX integrated very useful LISP functionalities into MAX's interactive environment such as massive list processing and music score representations. As their creators state, *bach* "takes advantage of Max's facilities for sound processing, real-time interaction and graphical programming, combining interactive writing and algorithmic control of symbolic musical material" ([Agostini and Ghisi, 2012], p. 374)

[4] https://www.youtube.com/watch?v=ynPWOMzossI

The *bach* library extends MAX facilities in regard to CAAC in several dimensions. First, it provides MAX with two new data types: rational numbers and Lisp-like linked lists or "llll". Rational numbers are fundamental for music symbolic representation. Fractions like 1/2, 3/8 represent rhythmic durations and relationships in the traditional notation system, and in several cases, float numbers (the data type to store these numbers in MAX) don't represent this rational relations accurately or conveniently for music notation.

The advantage of using llll's to represent big amounts of organized music information resides in how easily this data structures reflect the organization structure of symbolic representation of music. As the authors of this library state, Lisp-like linked lists were chosen because of the need to establish a data structure powerful enough to represent the complexity of a musical score, but flexible enough to be a generic data container suitable of being manipulated through a relatively small set of primitives [Agostini and Ghisi, 2012]. Thus, the large majority of *bach* objects were designed to perform operations upon llll's and exchange music scores represented in the form of specifically structured llls, a more efficient data type than MAX's regular lists. *bach* objects are able to perform basic operations such as retrieval of individual elements, iteration, reversal, sorting, splicing and merging in multiple and specific depth levels. The following examples add a more detailed explanation of the llll structure and use.

Figure 3.1 shows how llll's are structured and represented in a line of code (in this case, a MAX/MSP message). Both lists store six elements, but their categorization and grouping is very different, the dark-gray box in the left-upper corner represents a MAX/MSP message containing the lists. The parenthesis syntax is very characteristic of the LISP language and expresses the list's particular grouping and multi-level categorization.

Fig. 3.2 shows a simple application of an llll with musical information in the *bach* software. Two bach objects to display llll information are shown, these are the *bach.tree* and the *bach.score*. The tree representation shows how an llll is formatted to be correctly interpreted by the score object and how does the containing logic works (the color tags were added by me, for clarity). As it is shown, one voice (staff) contains measures which in turn contain measure info and notes, and every note container holds its duration (expressed in rational numbers), pitch and velocity information. Data shown in this example, however, are not the only information capable of being managed by *bach* objects. A note container for instance, can have a wide variety of data from display color to micro-tunings, articulations, expression marks, and much more. Other objects and properties of the *bach* library will be explained in the following sections, which is devoted to the design process of the system and the reflections generated through several loops of design according to the methodology description in section 1.4.

3.2.6 Real time and Interactive CAAC

The use of interactive music generation systems opened the possibility of speaking of composition in a meta level ([Rowe, 2001], Taube in [Guedes, 2008]). This imply the possibility of *design the composition methods* in systems capable of change their composition algorithms in response to a given input. As it is stated in [Rowe, 2001] by generating music based in improvised inputs the

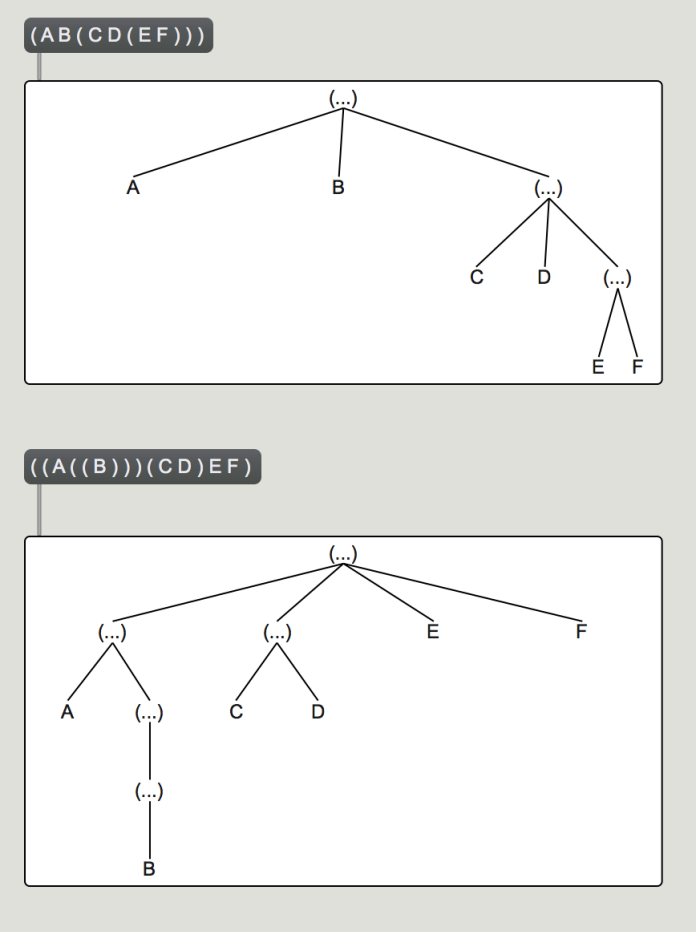


Figure 3.1: Two llll's with the same amount of elements but different structure

domains of music composition expand, and a meta-level appears in the process executed by the computer. I understood this as if the composer composed the way a machine compose. This what Guedes referred as real-time composition, and provides as example one of his works: *Etude for Unstable Time* (2003) a piece that was conceived to be "to be composed during a performance through the collaboration among three different entities: a dancer, a composer and a computer". Moreover, since the three entities would be composing over deriving processes, composition was occurring in a meta-compositional level.

More recently, it is possible to verify a growing interest in making CAAC systems more interactive and more real-time oriented. Exemplary works are those of the symbolic representation team at IRCAM like [Bresson et al., 2017] which describes how the last implementation of OpenMusic has "embedded features for the interactive and dynamic execution of musical processes, and for digital signal processing", traditionally, OM visual programs are evaluated on-demand by explicit request of the user, in refresh fashion, bit the latter implementation of this CAAC has implemented reactive outputs of visual program components to propagate notifications of change to downstream-connected components. Another example of this tendency can be seen in the work

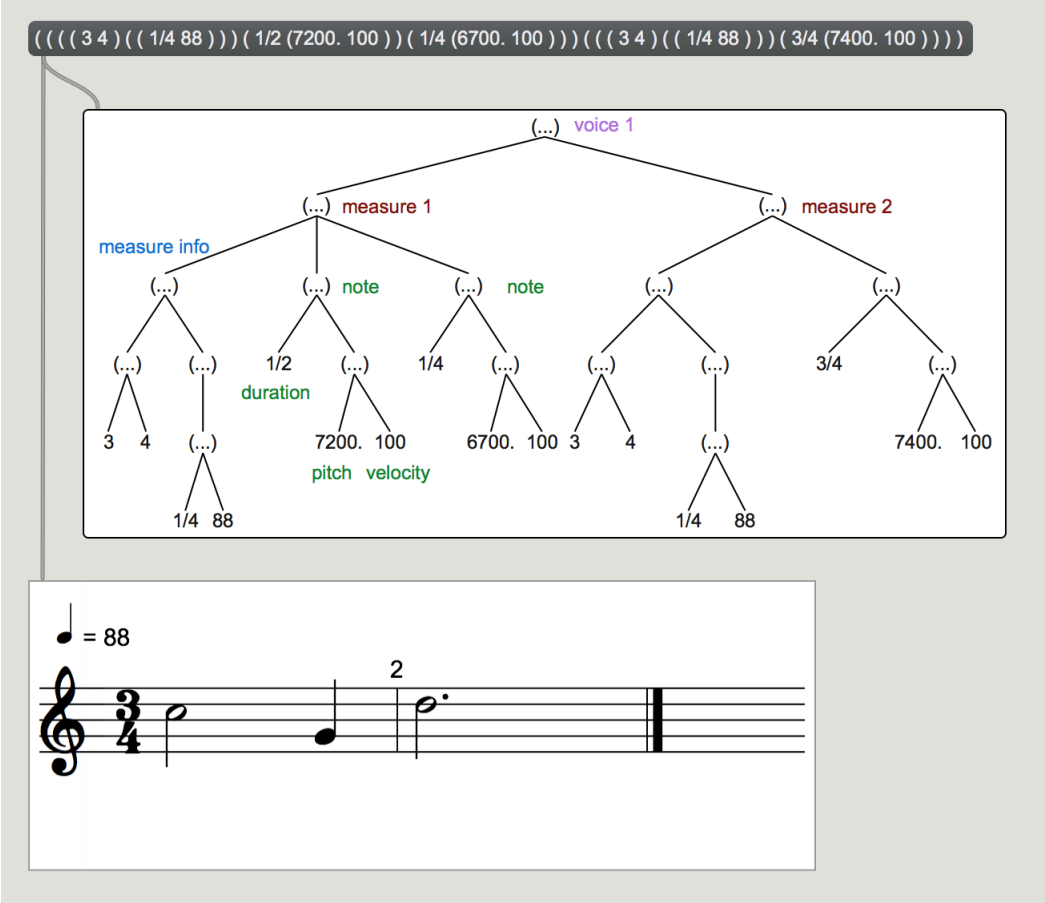


Figure 3.2: A typyical music representation in *bach* lists

of [Talbot et al., 2017], who implemented, also for OM, an score editor for interactive navigation throug a constraint solution space, motivated by their observation that "the existing constraint solvers often lack interactivity [...] the composers do not participate in the selection of a particular solution". Inserted in this tendencies, my tool enables a musician to interactively explore automatic generated variations of a previously performed music idea by playing on a musical instrument, as it is explained next.

3.3 Summary

Recent developments in computer assisted composition tend to emphasize real-time response and interactivity. Well established platforms like PWGL and OpenMusic required a composer to program an algorithm and wait to see or hear the results in differed time, but recent releases include libraries that allow to use some aspects of this programs in real-time interaction. Also, the *bach* library for MAX/MSP puts into a real-time responding music platform some procedures inherited from the LISP functionalities that PWL and OpenMusic have for computer assisted composition.

Chapter 4

The iterative design process of Moti-Var

Due to the exploratory methodology I adopted, the system experienced several stages of development. Each development stage included a goal to be achieved, a register of the implementations made, and a final evaluation and reflections that led to future sessions goals, all that registered in a log. At a given moment, a breakpoint in this loop was made to share the results obtained so far in this dissertation, so this chapter came mainly from the pages of that register. When rereading the log, I realized that presenting only the final solution will left aside important findings and insight that were as valuable as the final prototype.

This chapter begins with a general description of the expected characteristics of the system, and it is followed by five more sections dedicated to each stage of the development process. In each section, the goals, implementations and an evaluation of the results are presented.

4.1 General description of the system

One of the main objectives of Moti-Var is to favor hands-on improvisation and interactivity, taking distance from traditional off-line procedures in CAAC. Even though a visual interface was created to deliver relevant information, this work is intended to detach the user from watching the screen while generating variations. The general architecture of the system is shown in figure 4.1 an the expected work flow was conceived as follows:

1. The user records a musical gesture by playing notes in a MIDI keyboard. What s/he played should be stored in the computer's memory.

2. When recording stopped, the system extracts the contour information from the gesture and enters in "listening mode". This means the system is waiting for the user to input chords in the MIDI keyboard.

3. The system should be able to recognize when several pitches are played simultaneously and synthesize all pitch information as a PCset.

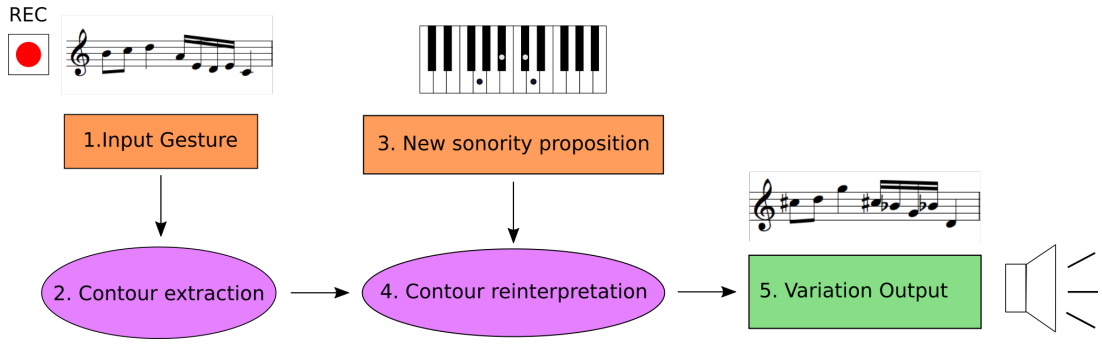


Figure 4.1: Moti-Var general architecture and work flow.

4. Every new input of a sonority triggers the generation of a new variation, so the previously recorder contour is reinterpreted accordingly.

5. The new variation is played and displayed in conventional music notation. If the user likes what s/he hears, there should be a way to store or save the variation generated by the computer.

After establishing that technologies such as arpeggiators and automatic accompaniment systems(see section 1.2) are too limited to achieve the purposes of this work, it was clear that a CAAC tool such as the *bach* library in MAX was appropriate to design a functional user interface and to implement the symbolic music processes needed. This tool offered a smooth integration between interactivity and symbolic processing of music information. My preliminary approach was to look for tools in the library that seemed to accomplish the task needed. I was aware of the *cage* collection of *bach* abstractions by the same authors, which is intended to "perform higher-level tasks, with a compositional rather than strictly technical connotation (e.g. melodic material generation, or computation of symbolic frequency modulation)"[1].

To decide if I was to use that tools, I made a preliminary exploration of the possibilities that both the *bach* and *cage* libraries offered for musical contour reinterpretation with pitch class set restrictions. Among all *cage* objects, *cage.profile.snap* provided a possible solution for the problem, so a simple patch like that in figure 4.2 was used to test the object in an interactive context. Given a profile composed by a series of MIDI pitches, this object approximates their pitch values to a predefined pitch grid. In this example, the pitch grid consists of all members of a given pitch class set [1, 2, 5] in the whole range of MIDI notes (0 to 127). The MIDI pitch profile is the whole set of pitches read from a *bach.roll* object. This object provides an spatial representation of rhythm, and it was preferred over *bach.score* since the system should make no assumptions regarding the tempo or metrics of the input gesture. The *bach.roll* object stores rhythmic information in the form of onset and duration time in milliseconds for each note. That information was left untouched when generating the variations, which appeared in a second *bach.roll* object.

[1] http://www.bachproject.net/cage/

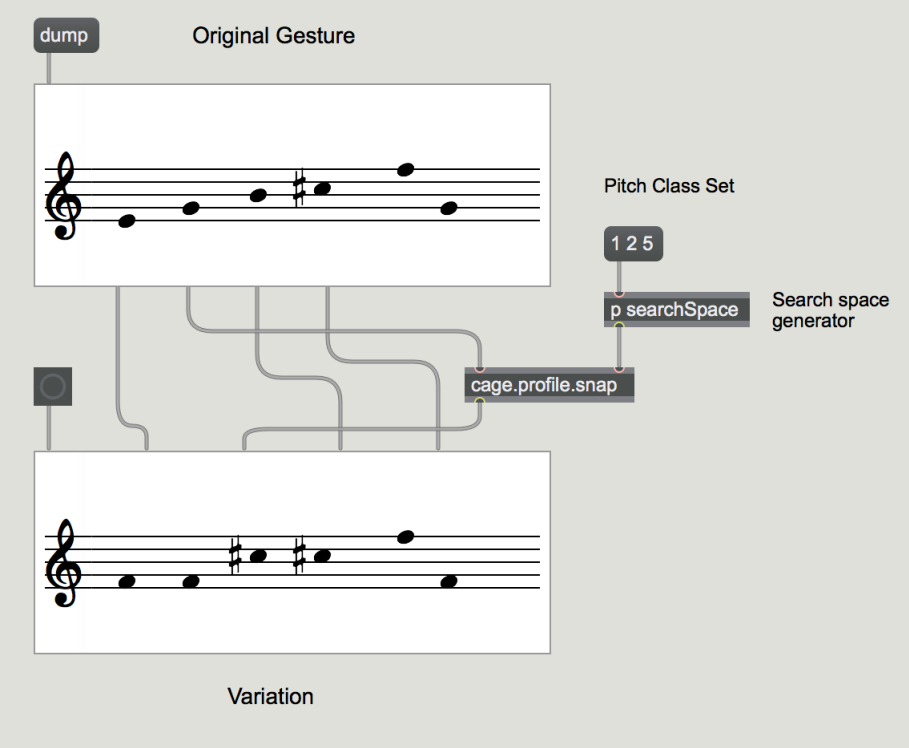


Figure 4.2: A first test with the *cage.profile.snap* object

In this example, the resulting contour has a clear resemblance to the original. Nevertheless, when two different pitches are close enough to one of the values of the pitch grid, they are approximated to the same value. Consequently, the resulting repeated notes destroy that particular contour segment's direction. Besides that, pitch class [2] is never used, so the variation will not reflect the aural qualities of the test PCset accurately. Though this test was not a solution to the problem, it showed that the *bach* library was an adequate technology for developing the prototypes, as the following sections account for.

4.2 Prototype 1

Since the sole approximations of pitch values was not sufficient to accurately represent musical contour with PCset restrictions, I thought that the music contour should be represented in a more accurate way and a PCset utilization constraint should be used to rebuild or reinterpret that contour in the variation.

Objective: Design an iterative interval approximation algorithm with an exhaustive PCset utilization constraint.

Implementation: An algorithm of this kind could be designed as a constraint solver. Given a limited set of variables and a limited set of constraints, a constraint solving algorithm looks for values in the variables that satisfy the constraints. Those constraints are expressed as functions that evaluate true or false. The *bach* library includes an object that solves constraint satisfaction

problems. In the case of this problem, the finite set of variables corresponded to the pitches of the notes present in the musical gesture, and the set of constraints would be the limitations of possible values that pitches can assume due to a pitch class set restriction, and how this pitch distribution produce a resemblance to the original contour. At the moment, establishing the variable set seemed trivial, but setting up the restrictions properly would demand an expertise in computer science and mathematics that I don't have, so I opted for another solving procedure.

My approach was to represent the music contour as a series of ordered intervals (see section 2.1.2) and perform individual approximations. For each note that is approximated, the next approximation target is obtained by summing the next ordered interval to the selected previously approximated note. All members of every pitch class that has been already used are removed from the search space. Once all pitch classes from the proposed PCset are exhausted, a new search space is generated as long as there are notes left in the gesture. I give this procedure the name of Adaptive Interval Approximation (AIA) and is detailed in the following pseudo-algorithm:

step 1 : Create a sequence of all the MIDI pitches in the original gesture as they appeared in time. (In this case, the example is in figure 4.4)

step 2 : Generate a list of ordered intervals that is the semitone differences between consecutive MIDI pitch values. The contour of the gesture in this example is: (+3 +4 +2 +4 -10), the sign is the direction of the interval.

step 3 : Create a search space containing all members of the proposed pitch class set. This is similar to the pitch grid used in the test with the *cage.profile.snap* object, but all members of the same pitch class should be grouped. That would be ((1 13 25 37 49 61 73 85 97 109 121) (2 14 26 38 50 62 74 86 98 110 122) (5 17 29 41 53 65 77 89 101 113 125)) for this example.

step 4 : Start the variation with the note in the search space that is the nearest to the first note of the original gesture.

step 5 : Determine the pitch class of the first note of the variation and remove all the notes belonging to the same pitch class in the search space.

step 6 : Add the first ordered interval to the first note in the variation.

step 7 : Pick the nearest note to that value from the remaining pitches in the search space

step 8 : Remove all members of that same pitch class from the search space.

step 9 : Add the following ordered interval to the last chosen note in the variation.

step 10 : Pick the nearest note from the remaining search space.

step 11 : Repeat until exhausting all ordered intervals. The search space should be restarted whenever it gets empty. Figure 4.3 shows a graphic explanation of the procedure and figure 4.4 shows the results of this example in a MAX patch.

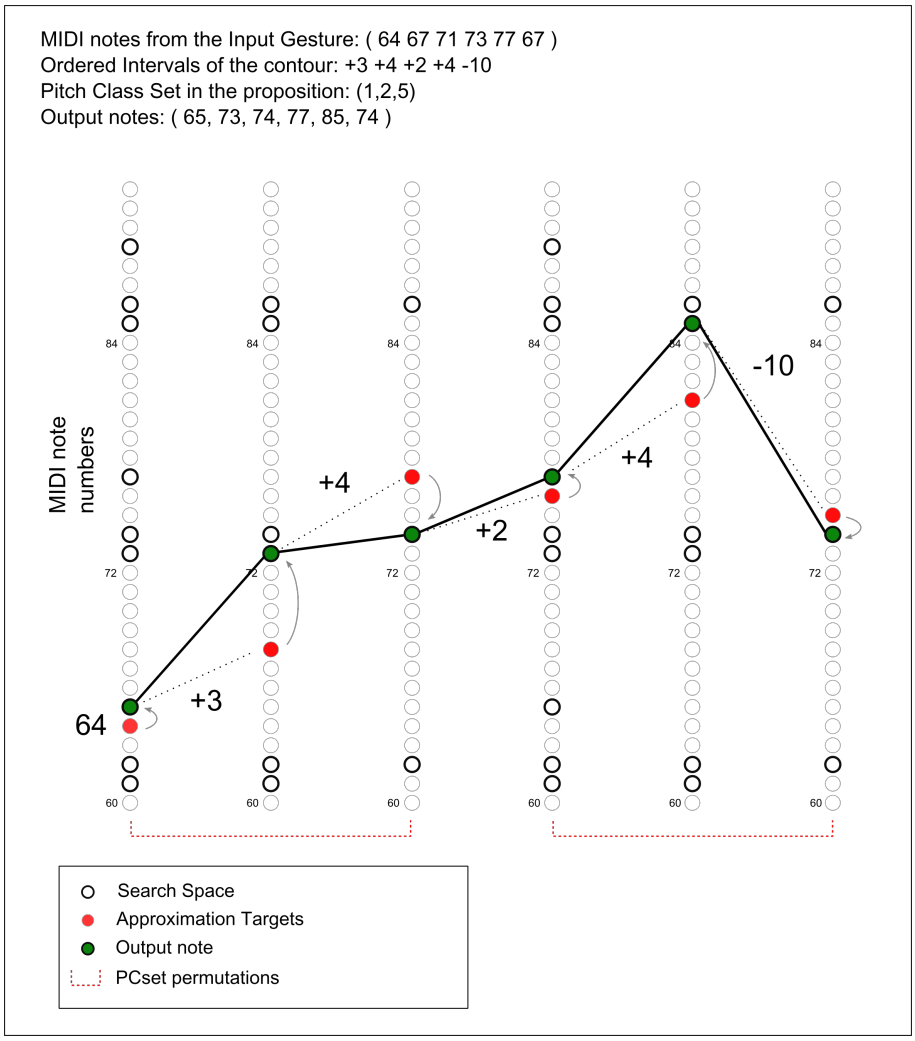


Figure 4.3: Graphic illustration of the Adaptive Interval Approximation

Evaluation and discussion: Provided that the original gesture had more or the same amount of pitches than the PCset proposition, the resulting variation will be PCset exhaustive, that is, all pitch classes in PCset proposition appeared at least once in the variation. A variation generated with this procedure will contain n/p permutations of the proposed PCset, being n the number of pitches in the gesture and p the quantity of pitches of the PCset. The algorithm also produced quite acceptable results in replicating the gesture contour, so I decided it was time to include it in an interactive context to deepen into its possibilities and better evaluate its behavior.

4.3 Prototype 2

This prototype had a user interface that allows to try different kinds of inputs: both input gestures and PCset propositions.

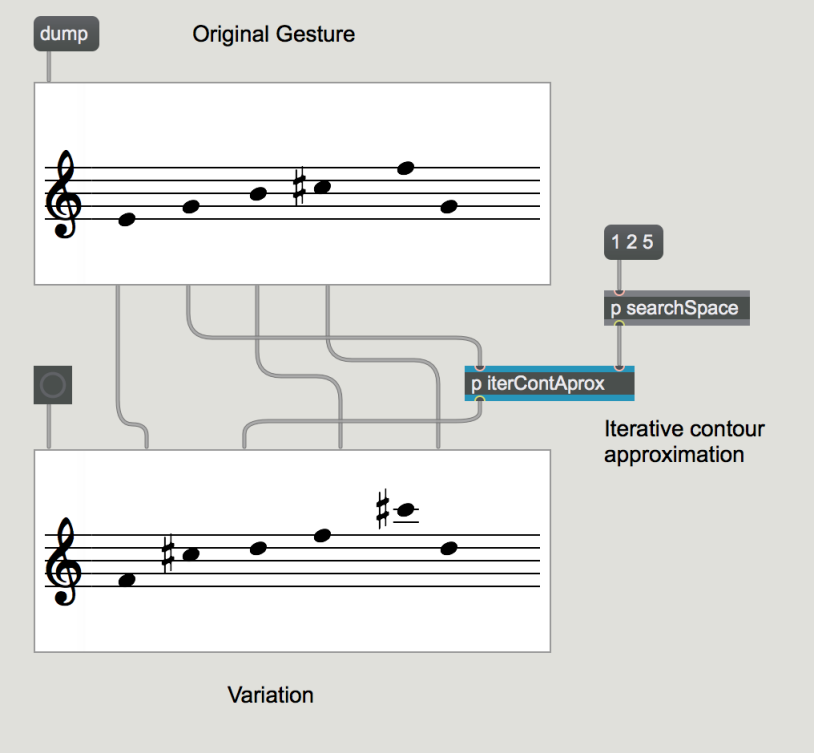


Figure 4.4: Results of the adaptive contour approximation process

Objective: Test the algorithm in an interactive setup that allows a user to record a gesture playing in a MIDI keyboard, and later input PCset proposals that trigger variations generation. The system should have also a storing mechanism that allows the user to store the variations when s/he likes them.

Implementation The *bach.roll* object has a quantity of methods that solved this necessities by receiving a simple MAX message in its main input. Besides that, by connecting a *bach.ezmidplay* abstraction in one of the outputs, a MIDI playback of the contents can be easily generated. The *bach.ezmidplay* abstraction was modified by me so it can receive information from two selectors that were included for choosing the desired MIDI input and output device. An abstraction called *bach.transcribe* takes care of transcribing a MIDI stream to llll so it can be read by the *bach.roll* object. To record, the user clicks on a toggle button, and the systems starts the transcription as soon as user plays the first note. When s/he is done recording, clicks again in the toggle to finish and the system retrieves the pitch information and calculate the ordered intervals in the contour.

Once recording is finished, the system waits for sonority proposals and a new variation is generated whenever the user plays at least two simultaneous notes in the MIDI keyboard. The sound module used should be properly set to receive MIDI information only from MAX and not directly from the keyboard. When a user plays a chord, what is meant to be heard are the variations triggered by the user's playing, not the actual notes s/he played in the keyboard. To recognize several notes as a chord, the system groups every note separated from the previous one

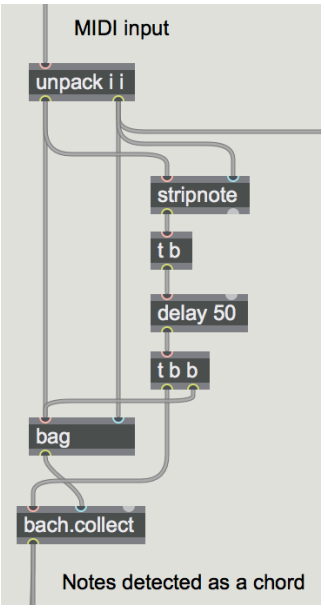


Figure 4.5: Chord detecting abstraction

by a maximum of 50 milliseconds, in a single unit, using the abstraction shown in figure 4.5. When a chord is recognized, a PCset is synthesized from all the pitches and used to build a new search space for generating a new variation. This instantaneous generation of variations increased the sensation of hands-on improvisation and allows the user to detach from the computer screen once the gesture is recorded. The interface's appearance can be seen in figure 4.6.

I considered that a user having a whole MIDI keyboard to control the variations would expect to have different versions of the gesture for different inputs in the keyboard, even if they correspond to the same PCset. So, to increase the responsiveness of the system, a modification in the algorithm was made so the lowest note in the user proposal was the first pitch class to start a variation. Additionally, I thought it would be stimulating that the proposal's range extension influenced the range of the variation. For that reason, when generating the variation, all contour intervals in the gesture were multiplied by p/o, where p is the proposal range in semitones and o is the range of the gesture's. This way, all intervals of the contour stretched proportionally.

Evaluation and discussion: Results with this functionalities were very interesting. Since rhythm is preserved in the variations, the similarity with the original is absolutely clear, even though the contour was not always exactly replicated. Moreover, the unexpected modifications of contour and the pitch class variety made up for surprisingly provoking variations. Let's take a closer look to this in a more detailed discussion around the algorithm's outcome.

Figure 4.7 shows a simple contour input to be reinterpreted according to four different sonority propositions as they were played in the keyboard[2]. Generally speaking, variations reflect accurately the original contour, except for variation number 3. In that case, a more constrained range

[2]To facilitate comparison and reading, all examples were transcribed to standard notation rather than presented from the system's interface and note chords from the inputs were written separated.

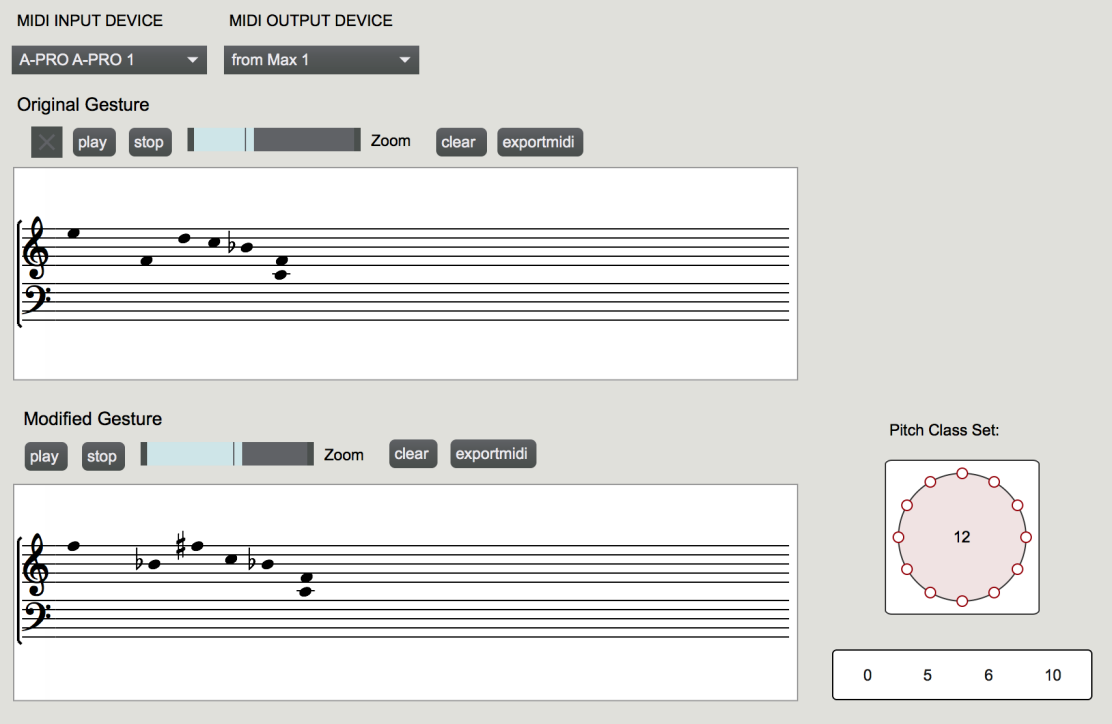


Figure 4.6: Moti-Var first prototype user interface in MAX.

interacted with pitch class restrictions causing the note selection to jump untidy. Since different dispositions of the same chord produced different variations, an expectation for a match between range in the keyboard and range in the variation emerged. E.g. one can expect that the same PCset proposition in different octaves could produce octave transpositions of the same variation. Moreover, mismatching ranges like those in variation number 4 happened to be confusing because the relationship between input and output is not straight forward.

Next test (figure 4.8) showed how in some situations a more exact replication of the contour happened to be desirable or expectable. In this test a very simple input contour having repeated notes and an implicit voice leading pattern was used. Because of the exhaustive PCset utilization restriction, those characteristics were never reflected in the variations. On the other hand, due to the exhaustive utilization of the PCset and the avoidance of pitch class repetitions, the variations have a richer aural qualities than the contour input, an unforeseen but attractive consequence of how the algorithm works. Range mismatch was even more evident in this test. For instance, despite proposition number 3 suggested a one semitone wider range than the model's, the variation spread exuberantly through two octaves.

The addition of the export MIDI function was a remarkable feature. Having several interesting variations in a few minutes was very easy. This experience made me think about an alternative usage for this algorithm: sometimes it can desirable to generate musically interesting results, rather than an exact replication of the contour. This behavior can be useful for a composer willing to "unlock" the potential of apparently banal or basic musical ideas, providing unsuspected paths

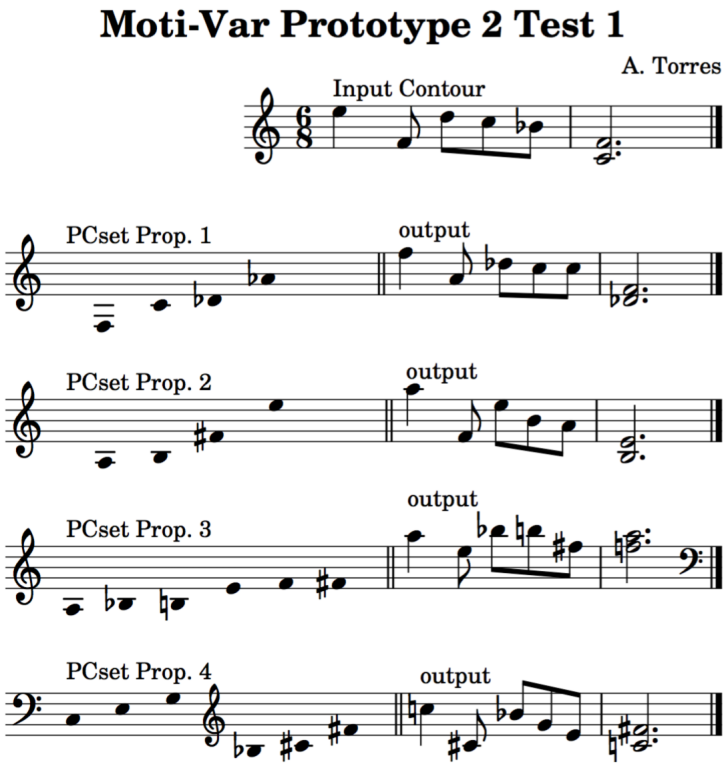


Figure 4.7: Prototype 2 transformations of a simple contour

for developing music material. This possibility remained unexplored until the final version of the prototype, but I will get back on this matter in the results discussion in chapter 4.

Despite this positive surprises, in other utilization contexts like pedagogical situations, a user may find the variation procedure puzzling. Besides the fore-mentioned range inconsistencies among the proposals and variations, virtually no repeated notes were maintained and implicit voice leading patterns are lost. Given that this characteristics can become salient when present in a musical idea, I thought it would be desirable to have a more strict imitation of the gesture's contour. I decided to try a more detailed representation of contour information.

4.4 Prototype 3

Objective: Improve the system behavior regarding:

- Correspondence between the range of sonority proposals in the keyboard and the variations the system generates.
- Identification and preservation of important contour elements like repeated notes and chords.
- Pitch class variety and exhaustive use of the PCset present in the proposition should be preserved.

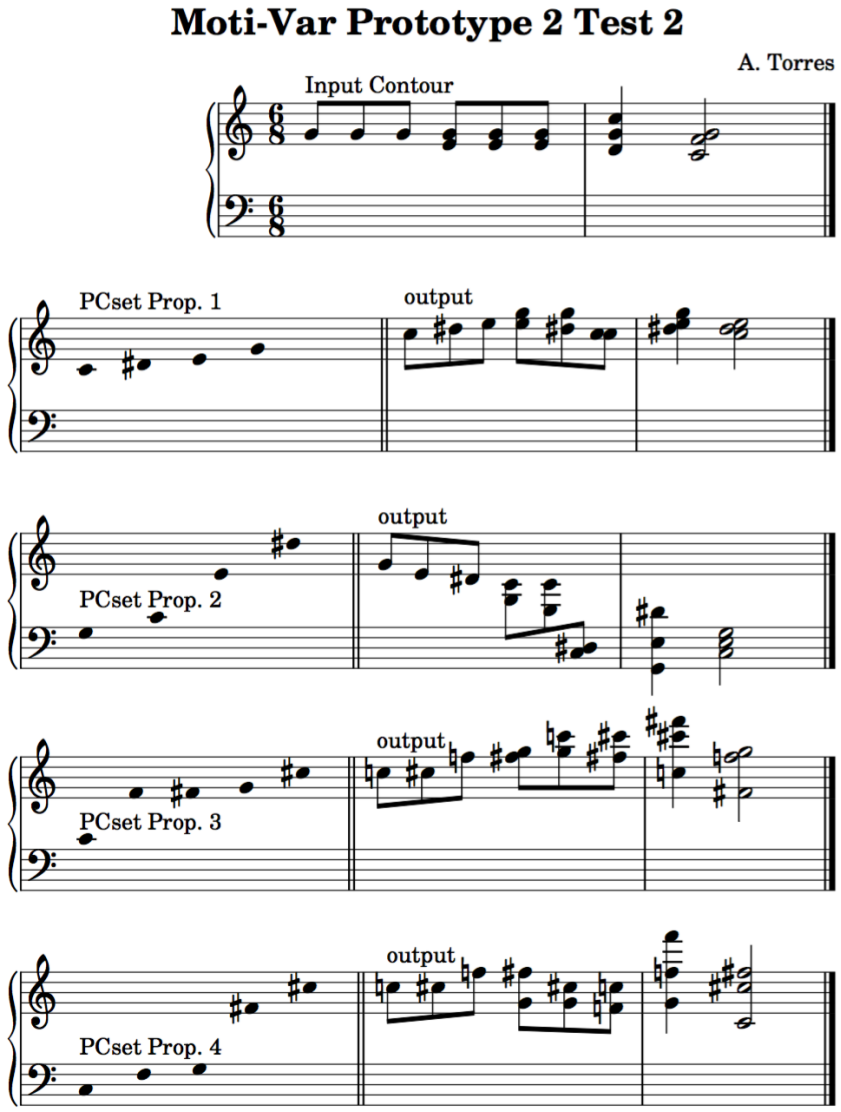


Figure 4.8: Prototype 2 transformations of a model with repeated notes and voice leading patterns

Implementation: It was necessary to find a way to identify repeated chords and notes and and keep the algorithm from changing them. I departed from representing a music contour as a set of points in a bi-dimensional space like in figure 2.2. Each point corresponded to each note in the input gesture, and was represented in the software by an (onset, MIDI pitch) pair. That information was grouped following a chunking model that is illustrated in figure 4.9. This way, redundant pitch information is discarded so repeated chords and notes are stored as chunks with a single pitch element and several onsets. Using bach's LISP like linked lists (llll) flexible structure, chunks had different formatting that clearly indicated four types of chunks as follows:

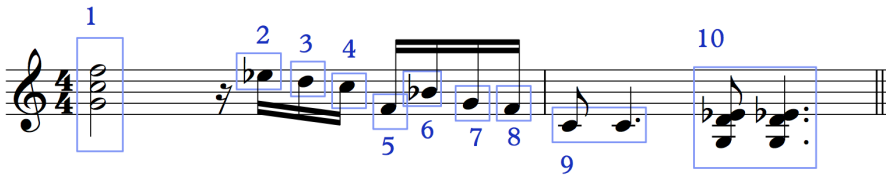


Figure 4.9: Example of how the chunking is made in a musical idea.

Single note	$(((pitch))((onset)))$
Repeated note	$(((pitch))((onset_1)...(onset_n))$
Single chord	$(((pitch_1...pitch_n))((onset_1...onset_n)))$
Repeated chord	$(((pitch_1...pitch_n))((onset_1...onset_n)...(onset_1...onset_n)))$

Later, the whole input contour was stored as a series of chunks, in a llll formatted as follows:

$$(chunk_1)(chunk_2)...(chunk_n)$$

For dealing with range transformations in the variations, a normalized pitch representation was more practical. All pitch information was reduced to the range of [0, 1], being 0 the lowest pitch in the gesture and 1 the highest[3]. The rest of pitches will take a proportional value of:

$$\frac{1}{h-l}(N-l)$$

where N is the MIDI pitch value of a given note and h and l the MIDI pitch number of the highest and the lowest pitches in the contour.

As it was made in the first prototype, normalized pitch intervals will substitute the pitch information for every note but the first one. In the case of chord chunks, differences are calculated in two directions: from one chunk to another, difference is always calculated between the highest notes in a chunk either chord or not. Inside every chord chunk, for the rest of the notes, differences are calculated in relation to the next highest pitch above. The final list of contour chunks of the example in figure 4.9 was like this:

[3] At this time I had not read the article of [Schultz, 2016] about normalized contour representation, to which I will refer later.

Chunk	**llll**
1	(((1. -0.227273 -0.227273)) ((1.068902 0.801771 0.639482)))
2	(((-0.090909)) ((1695.672607)))
3	(((-0.045455)) ((1888.222046)))
4	(((-0.090909)) ((2078.071777)))
5	(((-0.318182)) ((2269.165771)))
6	(((0.227273)) ((2425.781494)))
7	(((-0.136364)) ((2631.174316)))
8	(((-0.090909)) ((2791.760742)))
9	(((-0.227273)) ((2973.003418) (3346.415039)))
10	(((0.136364 -0.045455 -0.318182)) ((4512.428223 4518.831543 4504.288086) (4894.187988 4904.703613 4904.460938)))

For generating a variation, a similar algorithm to that in prototypes one and two was employed with slight modifications. For the variations match the range of the sonority proposal, the search space was limited between the lower and higher notes of the sonority proposal. The distance between these limits become a multiplying factor for all normalized intervals. This way, all intervals in the contour are stretched accordingly to the sonority proposal range. During preliminary tests, I realized that sometimes a better contour replication was obtained beyond the search space limits. For that reason I kept the proposal's range as the multiplying factor for the normalized intervals, but extended the search space one octave below and above the proposals' range to have more flexibility.

It resulted very hard to formalize an algorithm that used the PCset exhaustively and produced accurate contour replications. Meeting both restrictions required an extensive set of looking back restrictions, and that made very difficult to formalize convenient and fast-to-compute restrictions. For that reason, a compromise between pitch variety and exact replication was made. To achieve this, pitch class variety constraints were limited only to consecutive pitch elements.

Evaluation: Examples of the variations generated by this algorithm are shown in figure 4.10. The correspondence between the proposals and the resulting variations range improved. The input contour of this examples has sufficient notes to offer opportunities for using all pitch classes of the proposals, in all examples every pitch class is used at least once. Nevertheless, it was clear the previous restrictions produced a more even distribution of pitch classes.

Initially, this algorithm stood for a satisfactory solution to the problem. Range correspondence was good enough, and it was possible to program a more restricted constraint to increase pitch class variety. The system proved useful both for tonal and atonal materials and one of the rare cases when it was not satisfactory was dealing with contours that had voice leading patterns, implicit or explicit e.g. consecutive chords with notes in common. This kind of patterns were totally unrecognizable to the contour chunking. At that moment, I thought that voice leading management will appear as future work in this dissertation, but when I was looking for a definition of musical contour I find a paper regarding contour theory in music that opened a new landscape of

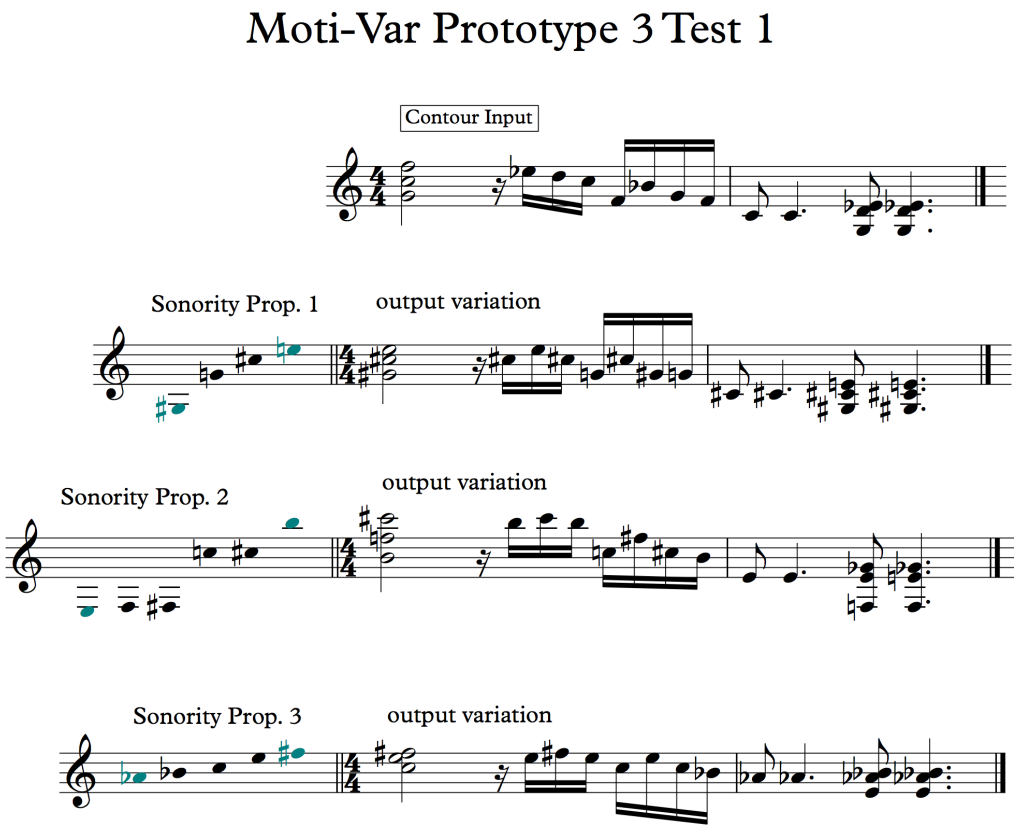


Figure 4.10: Variations employing contour chunking, normalized intervals and range restrictions in the search space.

possibilities and actually made that improvements possible. Next sections are about the application of contour theory in the algorithm of Moti-Var.

4.5 Prototype 4

Music contour representations from twentiethcentury music theory proved very useful to achieve two objectives in the development of Moti-Var: to generate variations that span exactly between the boundaries of a sonority proposal and to replicate accurately the voice leading patterns implicit in the contour input.

Objectives: To implement contour theory in the algorithm of Moti-Var and explore how that can contribute to an accurate replication of the original contour in the variations. To use contour theory to search an exact correspondence between the pitch range of the proposals and the variations.

Implementation: The application of contour theory made previous chunking unnecessary. A contour class (CC) representation by itself epitomizes repeated notes, repeated chords and voice leading patterns accurately. In this version of the algorithm, all MIDI pitch information in the

input gesture were substituted by contour pitch information. Since CC represent only the order relationship between all the different pitches present in the gesture, interval or semitone distances are ignored. Also, contour pitch information was normalized, (as in [Schultz, 2016]) and used later to approximate contour pitches in a search space as the following pseudo-algorithm explains:

step 1 : Create a normalized contour class (see section 2.2) of the input gesture; for every different pitch in the input contour exists a different normalized value (see figure 4.11)

step 2 : When a sonority proposal is entered, create a search space with all possible members of the pitch class set **between the proposal's range**.

step 3 : Pitches in the search space are numerated as if they were contour pitches of a normalized contour class.

step 4 : Substitute every normalized contour pitch in the input for the pitch that has the nearest **normalized contour value** in the search space. This guarantees that the lowest and highest pitches in the sonority proposal are always the lowest and highest in the variation.

step 5 : To generate a variation, every pitch of the input contour is substituted by its correspondent approximation in the search space.

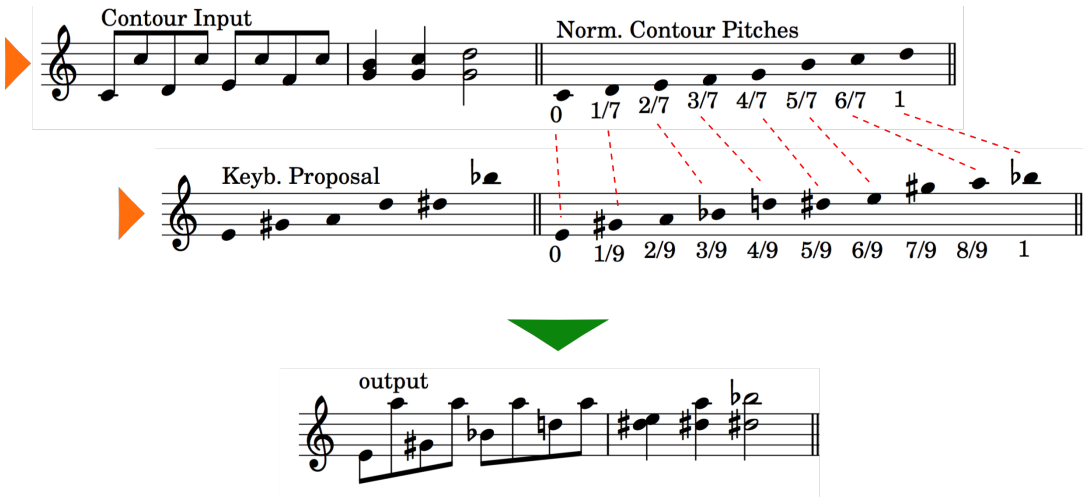


Figure 4.11: Normalized Contour Approximation (NCA).

Evaluation: This procedure guaranteed exact replication of repeated notes and voice leading patterns. Provided that the proposition have equal or greater number of contour pitches than the input gesture, a perfectly equivalent contour is obtained in the variation. Exhaustive use of pitch classes in the PCset was not assured, but the following tests showed a noticeable improvement in this matter when compared to the results of prototype 3.

The following test is a comparison among the results delivered by the algorithms of this prototype and prototype 3. Each result is labeled according to the name of the algorithm, Adaptive

Interval Approximation (AIA) for prototype 3, and Normalized Contour Approximation (NCA) for prototype 4. The results of both procedures were compared to a variation written by myself, in the understanding that the more similar a variation is to a human version, the more successful it is. All human variations were written according to the three goals that were pursued so far:

- An equivalent or almost equivalent replication of the contour profile of the input contour.
- An exhaustive utilization of the pitch classes from the sonority proposal.
- Exact correspondence between the ranges of the sonority proposal and the output variation.

Figure 4.12 shows a comparison among a human constructed variations and the output variations generated by NCA and AIA algorithms with the same sonority proposal. A simple melodic contour was the contour input for this tests, and the results for each sonority proposal are discussed below:

Proposal 1: This proposal implies a range reduction and one contour pitch less than the input. NCA managed to replicate the human proposal exactly, while the AIA result is slightly out of range. Since the proposal has less pitches than the original, exhaustive PCset utilization was granted.

Proposal 2: Both algorithms have been functional when the proposals have tonal implications, for that reason I included a test with a F minor triad. Such proposal generates a search space of 6 possible contour pitches having only three different pitch classes (two F minor triads in different consecutive octaves), therefore PCset exhaustive utilization is not guaranteed but very likely to occur. In this example, NCA successfully replicate the original contour but with a different pitch selection than the human variation. AIA has a more accurate version of the original intervals and is even tonally correct but overrode the differences of height among the third and fifth pitches of the contour.

Proposal 3: Without a strict PCset exhausting restriction, it is not guaranteed that the variations include all pitch classes in the proposal. The addition of one pitch class to the proposal was not accurately reflected in the variations. This case evidences the compromise that was made among using all PCset members and replicating the contour between range restrictions. A noticeable flaw of AIA here was that generated the same variation to proposals 2 and three, while NCA got to include the E that was added, but omitted the A flat. When comparing those variations to the expected result (human variation), I realized that a proper formalization of the music procedure was still missing to accomplish the three objectives I mentioned before.

Proposals 4 and 5: These examples only confirmed the necessity of a more functional pitch class restriction. Both proposals generate search spaces with multiple members of the pitch classes. What made evident that the algorithms are not PCset exhaustive is that all variations included pitch class repetitions in response to proposals that had more pitch class variety.

Valuable insight was obtained from analyzing these results. To approximate the input contour using exhaustive PCset restrictions was simple, and was the point of departure of this design

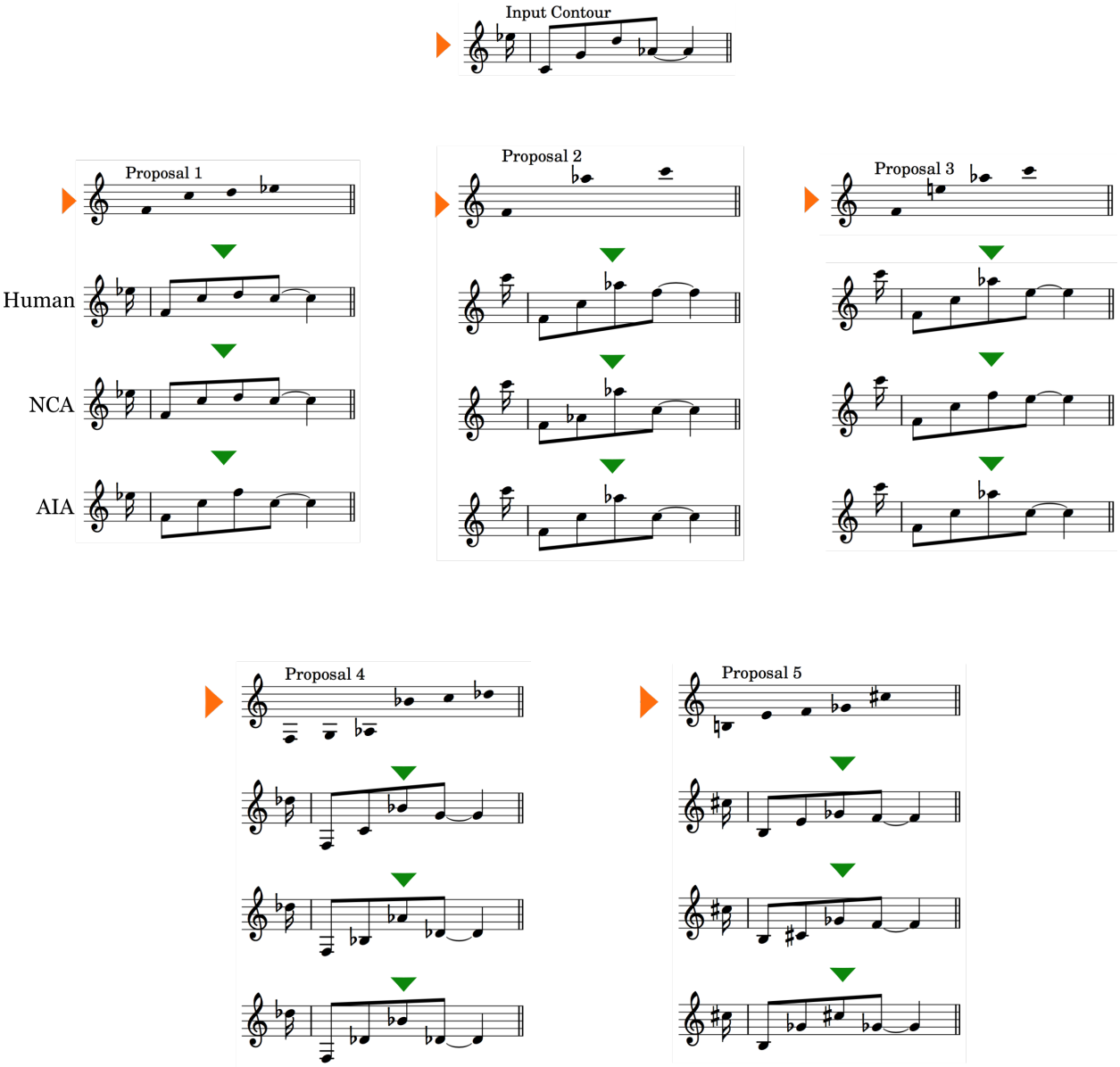


Figure 4.12: Comparison between human variation, normalized contour approximation (NCA) and adaptive interval approximation (AIA).

process. Since this is an exploratory work, it was totally worth to deepen into the problem implications, going as further as possible. The tests with human comparison shown that restrictions of range and a more detailed replication of particular voice leading patterns and repeated notes increased the difficulty of formalizing the problem properly.

That proper formalization was not possible to accomplish in the time span of this dissertation. Nevertheless, in order to present a satisfactory functional prototype, I managed to integrate together some features of adaptive approximation and contour theory in the last prototype of Moti-Var, which is presented in the following section.

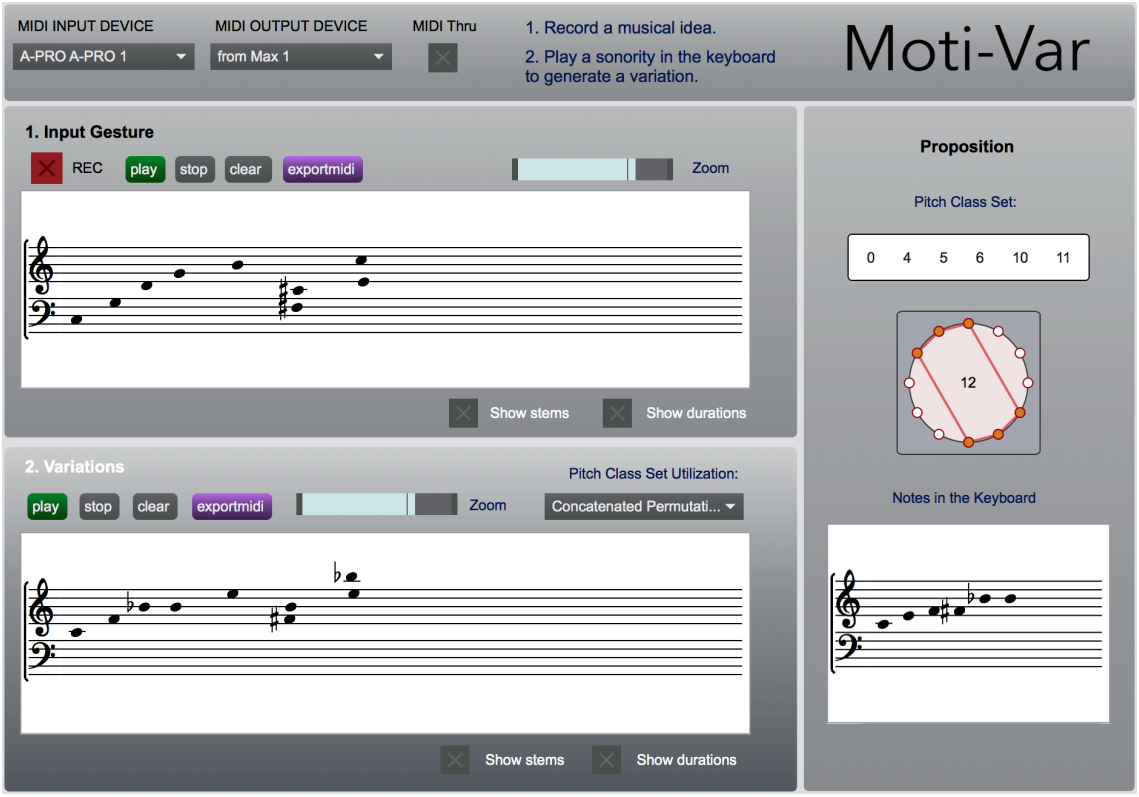


Figure 4.13: Final version of the Moti-Var user interface.

4.6 Prototype 5

The aural qualities of the variations made with the first algorithms (that had PCset exhaustive use restrictions) were never surpassed by that ones with a more precise contour imitation and range restrictions. The most aurally interesting variations were obtained when pitch class variety was assured, and I wanted to bring back that characteristics to the final prototype. By sacrificing a little precision in range matching and opting for different contour imitation restrictions, this last prototype addressed several issues that have emerged during the process and integrated many findings obtained in the previous developmental phases.

Implementation: Two algorithms were implemented in this prototype, but first, I want to present a short description of the improvements that were made in the user interface, which last version is shown in figure 4.13. The interface was divided in four sections. The upper transversal section deals with MIDI configuration, the two major blocks at the left show the contour input and variation spaces and at the right there is a block that delivers information about the sonority propositions.

Besides the MIDI devices selectors that appeared before, a MIDI Thru toggle was included in the MIDI controls. When entering sonority proposals in the system, the actual notes of the keyboard are not heard, but the variation is played instantly. During informal tests with colleagues and myself, sometimes it was necessary to suspend the variation generation for trying new ideas

in the keyboard, or even our own improvements to the variations. Before this addition, that was not possible because the actual keyboard playing was only heard in record mode.

In this prototype, variations can be generated using two different procedures. A list selector in the variation section was included to choose among the two methods. That variation procedures will be explained later.

The proposition information section included a circular graphic display and a *bach.tree* object to show the PCset of the sonority proposal. A two staff system was added to display the actual notes played by the user in conventional music notation.

The two different procedures for generation the variations in this prototype have their own particularities. One replicates exactly the contour of the input gesture and the other one assures the the biggest pitch class variety possible. I will start explaining the first algorithm.

Filling the contour pitch space with PCset members: This procedure establishes a simple correspondence between a search space of pitches and the CC integers from the input gesture. As long as the gesture has equal or greater quantity of CC integers than pitch classes in the PCset, all pitch classes in the proposal will appear at least once. Repeated notes and voice leading patterns are preserved. This is the pseudo-algorithm that explains the procedure:

step 1 : Create a not normalized contour class (CC) of the input gesture.

step 2 : When a new sonority proposal is entered, create a search space with all members of its PCset in the MIDI range.

step 3 : Locate in the search space the lowest pitch of the sonority proposal and label it as CC integer 0. Departing from that pitch, assign the rest of CC integers in the gesture to the following higher notes in the search space until using all CC integers.

step 4 : To create the variation, substitute every CC integer in the input gesture by its corresponding MIDI pitch in the search space. A graphic explanation of the procedure can be seen in figure 4.14

Since it is based on contour representation, this procedure replicates exactly any repeated notes and voice leading patterns. It ignores completely the sonority proposal range extension, but at least the lowest note of the propositions will be the lowest note in the variation, assuring a range correspondence between the proposal and the variation (e.g. the same sonority proposal in different octaves produces transpositions of the same variation). The total pitch extension of the variations depends on how many different contour pitches the input gesture has, and the size of the intervals between the members of the PCset. Transformations in the extension of intervals were discarded so far, since it was my appreciation that a more accurate contour replication is aesthetically more relevant, as well as having in the variations always the same cardinality of

Figure 4.14: Variations generated by prototype 5 using the search space as pitch contour space.

contour pitches than the input gesture has[4]. The next algorithm assures even more pitch class variety but with a very loose interpretation of the contour.

Contour Interval Guided Walk (CIGW): To generate a variation, this algorithm uses contour intervals to walk through a search space that gradually eliminates already used pitch class sets, exhausting cyclically all PCset members[5]. This is very similar to the AIA explained in section 4.2. The difference is that instead of semitone intervals, the input gesture is represented as a Contour Interval Succession (CIS)[6] as it was considered a simpler representation of changes of pitch over time. Moreover, the PCset restriction was refined to avoid pitch class repetitions between the last and the first notes of consecutive PCset permutations. The whole procedure is explained in the

[4]E.g. In prototypes 3 and 4, it was possible to attempt to replicate a 7 contour pitches gesture with only two different pitches, it is my appreciation that this is somewhat problematic. In this prototype, PCset members are added as much as different CC integers exist in the input gesture.

[5]As in prototype 2, a variation will include n/p permutations of the PCset, where n is the number of pitches in the input gesture and p the quantity of pitches in the PCset

[6]See section 2.2 for an explanation of CIS.

following pseudo-algorithm and in figure 4.15:

step 1 : Create a CIS representation of the input gesture, including the first CC integer as if it was a contour interval from CC integer 0.

step 2 : When a new sonority proposal is entered, create a search space with all members of its PCset in the MIDI range.

step 3 : Locate in the search space the lowest pitch of the sonority proposal and assume it is CC integer 0.

step 4 : Pick the first note of the variation by jumping from the CC 0 note of the previous step. Jump as many **pitch class set members** in the MIDI pitch range as the first contour interval of the CIS.

step 5 : From the search space, remove all members of the pitch class of the first note.

step 6 : From the first note, jump as many **pitch class set members** in the MIDI pitch range as the second contour interval of the CIS.

step 7 : In the updated search space, find the nearest available pitch.

step 8 : Remove all members of that same pitch class in the search space.

step 9 : From the second note, jump as many pitch class set members as the second contour interval of the CIS.

step 10 : In the updated search space, find the nearest available note.

step 11 : Continue the same until iterating all contour intervals in CIS.

step 12 : Once the search space is exhausted, create a new one including all PCset members in the MIDI register **but** the last pitch class used. Allow the last note of a previous permutation to appear in the second search space of the next permutation.

A single code was used to execute both algorithms in the system. The input gesture is stored as a series of (onset, CC integer) pairs, where all notes belonging to a chord (separated less than 50ms) are grouped in a single chunk and sorted from highest to the lowest. When the user select one of the algorithms, the data is sent to different objects that do the proper calculations. This way, a single contour representation is used for both procedures. The code (MAX objects) is shown in figure 4.16

Evaluation:

One of the more satisfying accomplishments of this prototype was to have again variations with a rich pitch class variety, though each algorithm achieve this by different means and with different results. The distinctive features of each algorithm are analyzed in the following commented examples:

Example 1: The sinuous profile of the input gesture is perfectly reproduced by the contour filling algorithm (fig. 4.17). An exhaustive use of the PCset is made, but patterns that include note pairs reiterations are strictly respected. The algorithm adds the necessary PCset members to have an equal amount of contour pitches than the input gesture.

In the variation generated by the CIGW, the PCset appears in the form of concatenated permutations that intend to replicate the input contour. Due to a more restricted PCset utilization, it is impossible in some cases to achieve a perfect contour replication. The profile in the variation descended exuberantly in an extended range of two octaves. I found this algorithm behavior fascinating and allowed me to improvise exploring the possible transformations of really simple motifs into fresh and distinct new ideas.

Example 2: A simple rhythmic motif with chords delivered also interesting results (fig. 4.18). The input gesture include a transposition of the same chord (F major). Because of the algorithm's contour representation, it makes no assumption about tonality, so the gesture is treated as having six different pitches. That is why in the variation the high and low chord have different pitch classes[7]. In turn, the CIGW produced a more dynamic and aurally richer variation of the gesture, making evident the aesthetic value of pitch variety, even though the chord repetition from the input was discarded.

4.7 Summary

The iterative design process of Moti-Var allowed to explore several aspects of automatic musical variation generation. Proficiency in writing variations of a musical idea is a fundamental skill for any good composer that can be approached in multiple manners. My approach was to replicate (or imitate) the musical contour (the profile of pitch changes in time) of a musical gesture according to PCset restrictions that are deduced from sonority proposals played in a MIDI keyboard.

All the solutions that were tested include some type of restricted approximation to an expected value. Different algorithms were tested during the iterative design process: simple pitch approximation to a pitch grid, adaptive interval approximation and approximation using normalized contour. The last system integrated implementations of music contour theory with adaptive approximation procedures from the first prototypes. The final prototype is able to reinterpret the same contour information in two ways: one replicates exactly the contour profile of the input gesture using as many pitch classes as necessary and the other one generates variations with much more pitch variety but within a more loose interpretation of the gesture profile.

Besides that, it can be said that the following features summarize what would be an ideal variation addressing the desirable restrictions that appeared in the process:

- The general contour of the variation should have a noticeable resemblance to that of the input gesture.

[7] Nevertheless, if a three note PCset was used, the transposed chord pattern replicates in the variation.

- The variation should preserve important contour elements like repeated notes and chords, and voice leading patterns.
- The variation should exhibit pitch class variety and an exhaustive use of the PCset when possible.
- The range of the variations should have an exact correspondence to the range of the proposals in the keyboard.

Though none of the prototypes achieve these goals perfectly, valuable insights were obtained about a music generation problem that has not been previously approached. A rich landscape of possibilities for future work in automatic music variation was another outcome if this exploration that is to be explained in the following chapter.

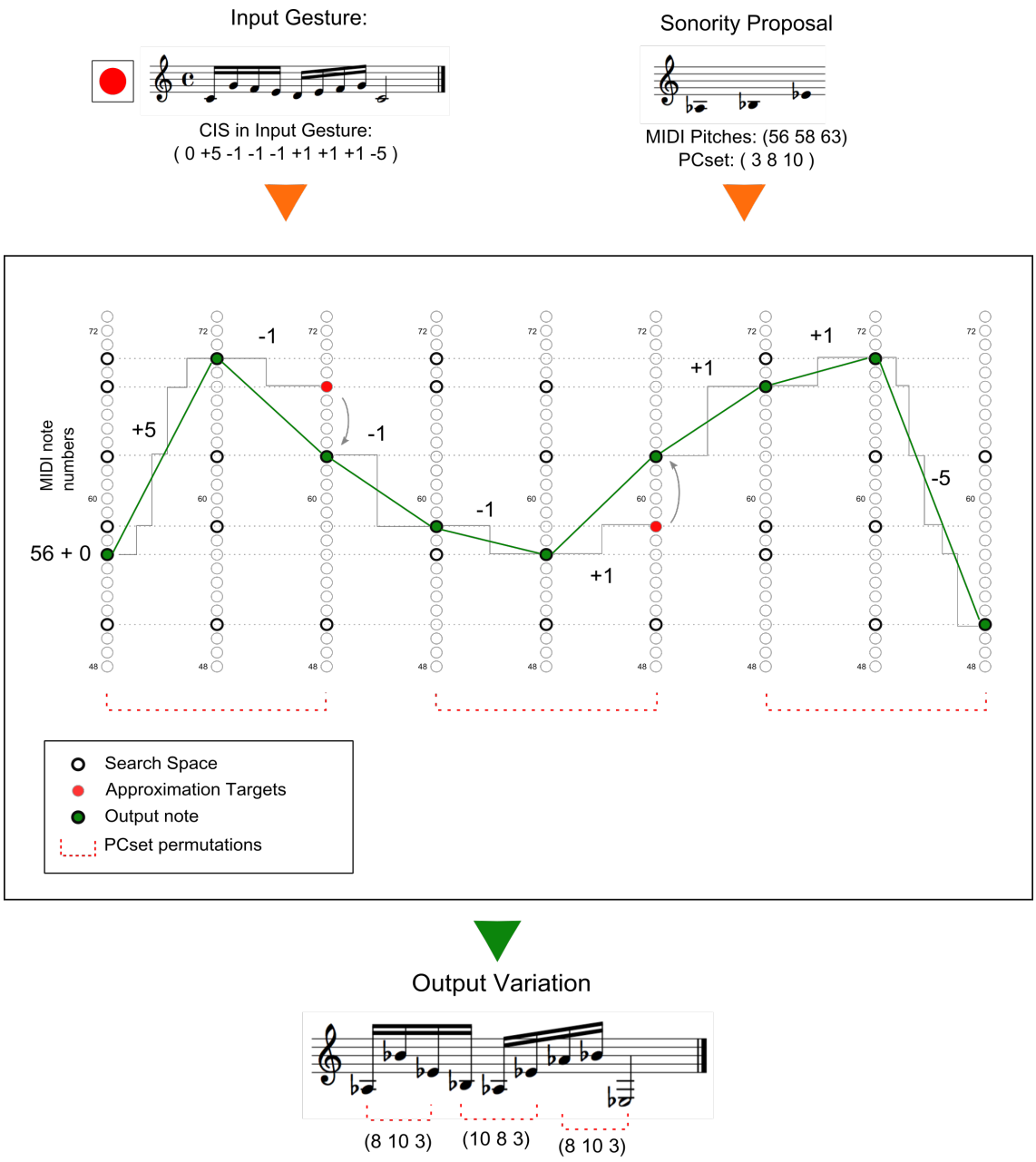


Figure 4.15: Graphic explanation of the CIGW procedure.

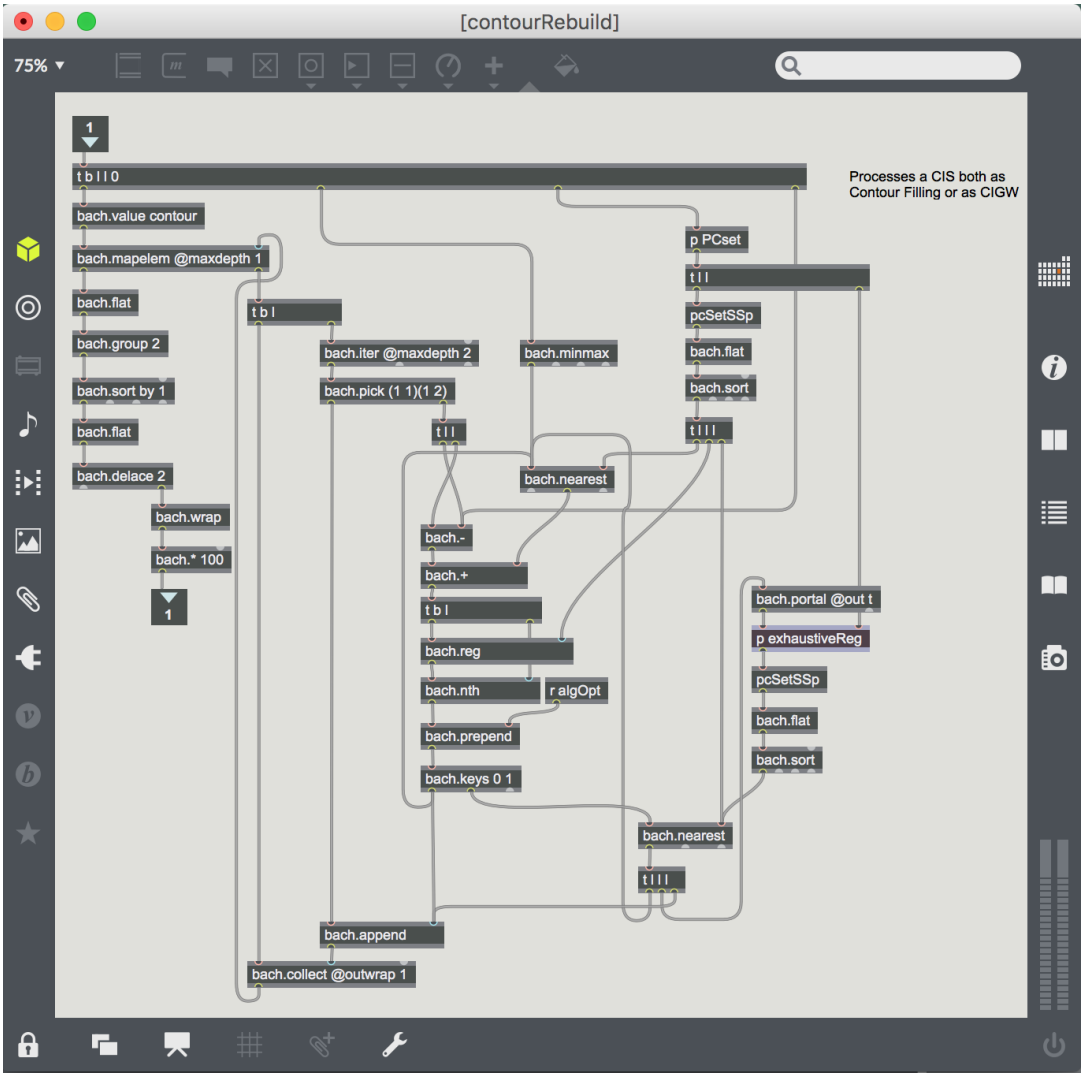


Figure 4.16: Abstraction in MAX that deals with the variation generation in Moti-Var.

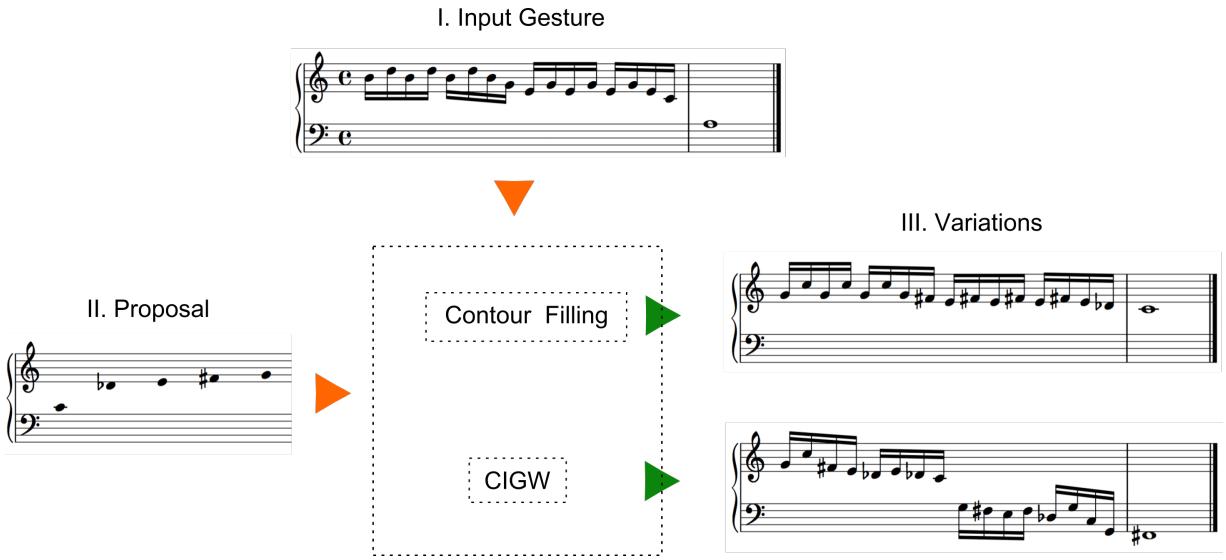


Figure 4.17: Transformations in Prototype 5 of a melodic idea.

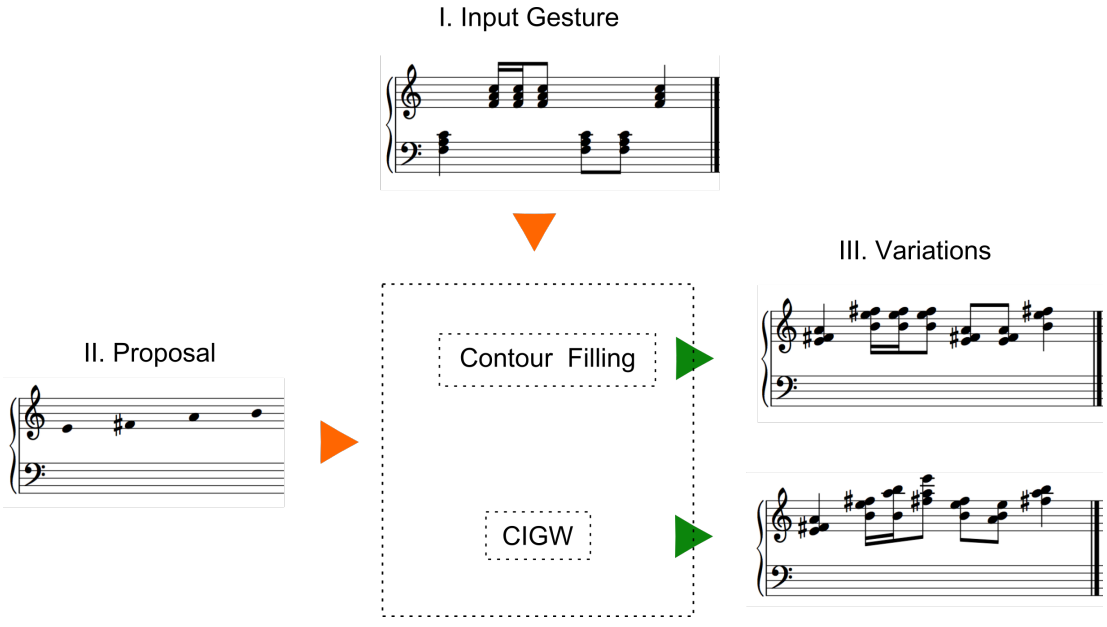


Figure 4.18: Transformations in Prototype 5 of a rhythmic chords idea.

Chapter 5

Conclusions and future work

5.1 Moti-Var and its context

Moti-Var is an application whose context is algorithmic composition, computer assisted composition and interactive music systems, integrating characteristics of those three practices. Computer assisted composition systems were traditionally off-line platforms that, after a refreshment command, process and compute musical information to deliver results in musical notation or using other types of music or sound representation. In the past decades, interactivity, real-time response, and online functioning were increasingly present in those applications ([Agostini and Ghisi, 2012], [Bresson et al., 2017], [Talbot et al., 2017]) as well as developed as characteristic features of automatic music generation systems that are intended to live music performance ([Rowe, 1993], [Winkler, 2001], [Guedes, 2008]). Among all this advances, François Pachet's Continuator [Pachet, 2002] is a remarkable work, not only for the achievement of modeling music style in real time, but also for exposing the fact that hearing oneself imitated by the computer is an intriguing and engaging experience. That background served as inspiration for the creation of Moti-Var.

As an attempt to overcome my own resistance to post-tonal music language, I decided to explore how to create a system that facilitate the use of improvisational methods of composition to aurally explore the characteristics of post-tonal macroharmonies. That system was devised as an automatic music variation generator, which after playing a musical gesture (focusing on rhythm and general pitch profile) was capable of reinterpret it with pitch class set restrictions deduced from a simple chord input in a MIDI keyboard. An iterative process of design and exploration was the methodology chosen, and after creating five different prototypes I came the following conclusions.

5.2 MIR for pitch contour

Music contour can be regarded as the changes of pitch in time. Though is not the only way of understanding it, that was the approach used in this work. In a more detailed manner, pitch contour was here represented as a series of points in a bi-dimensional space, being the *y* axis pitch

Algorithm	**Description**	**Results**
Search Space as Contour Pitch Space	CC integers are assigned to pitches in the search space: starting from the lowest note of the sonority proposal.	Very likely to be PCset exhaustive with precise contour replication including repetitions of chord and notes, and voice leading patterns. There is some correspondence among the ranges of proposal and variations.
CIGW	A walk through a search space guided by interval contours that adapt their departure points from iterative approximations.	A variable resemblance to the original contour. Assures the maximum pitch class variety by using concatenated permutations of the PCset

Table 5.1: A summary of the two algorithms present in prototype 5 of Moti-Var

and x axis time. In this work, pitch was represented either by a MIDI note number or by a CC integer. The latter was preferred since it is a more abstract representation, hence, is generalizable as a model to be reinterpreted or imitated with pitch class set restrictions. Contour pitches can be represented either as absolute CC integers or as a CIS, which represents contour distances between consecutive pitches of the gesture. Absolute or relative representation of contour pitch proved useful for different aesthetic purposes. Absolute representation facilitated a more exact replication of the input contour while relative representations allowed to generate more distant variations that were interesting as alternative developments of a musical idea.

5.3 Contour reinterpretation in real time

It takes several levels of formalization and a complex constraint definition to replicate the process a composer uses to reinterpret a music contour within the restrictions that emerged during this work. Contour theory proved very useful in the formalization of that procedure, and allowed to obtain very reasonable results within a very short computation time. Otherwise, it would have been necessary to invert massive computing resources by using brute force constraint solving algorithms[1].

Two alternatives to solve the problem of contour variation with PCset restrictions were implemented after integrating several issues that emerged during the process. Since both of them have valuable aesthetic characteristics, I opted for include the two of them in the final prototype. Their main characteristics are summarized in table 5.1

[1] In CSP, brute force algorithms are the ones that test absolutely all possible combinations of variable values from the search space in order to find the optimum solution. This usually implies long computation time, and the addition of new parameters produce an exponential increasing of the amount of solutions to test.

5.4 The aesthetic value of Moti-Var

This work allowed me to get closer to post-tonal music languages from a vivid and sensory perspective. Hearing my musical ideas transformed instantly, and acknowledging the aural qualities of post-tonal sonorities came to dissipate my previous resistance to use and accept this musical languages.

I felt very motivated to write music in this style due to the experience of hearing this sonorities in a musical context. As a consequence, my musical vocabulary increased, even developing preference for some particular sonorities. I considered this a step in the right direction for strengthening my singularity as a composer while extending my artistic horizons.

The inclusion of the two different algorithms helps Moti-Var to be a valuable tool for generating musical ideas to use in a composition, specially to "unlock" the potential of apparently simple musical motives (with the CIGW) and to instill different dramatic characters in a musical idea, by reinterpreting it with particular pitch combinations (Search Space as Contour Pitch Space)

5.5 Future Work

Future refinements and extensions to the tool can come from several activities where Moti-Var could be used:

- As a composition tool: composing a music piece using the tool will show what refinements are necessary to speed up the transition from the platform to an actual music score. In Moti-Var, rhythm is only represented as onset and duration time of the notes, not in actual music notation. A possible alternative would be to develop a version of Moti-Var as a plug-in for Sibelius, Finale and other music notation software.

- As a performing tool: since variations are generated after contour as a series of points in a bi-dimensional space, contour information can be received from other sources such as: musical contour extracted from an audio signal, motion capture devices or graphic displays, or kinetic musical interfaces. E.g. One musician can use a gestural interface with one hand while the other hand controls the pitch combination desired in a keyboard to explore kinetic music generation.

- The use of the tool by itself: Some refinement in the formalization of variation generation procedures can lead to a system that satisfies all the constraints that appeared during the work. Also, it would be interesting to include contour input manipulation from a visual perspective, so the contour also can be varied in a graphic display before testing more sonorities. Finally, by extending the user interface, complex or lengthy gestures could be interpreted using more than one PCset or sonority proposal.

I would like to conclude this dissertation emphasizing that due to this work, I finally overcome my resistance to post-tonal music languages. I am very satisfied of having developed a tool that I

considered useful not only for me, but for any one with the same resistance I had or that want to discover post tonal music from a more sensorial and richer perspective.

Appendix A

Time Pictures

This chapter includes a brief explanation of the composition process of "Time Pictures" a small piano suite that was written to evaluate the utilization of Moti-Var in assisted composition. Following this, two versions of the music score are offered, the second one indicates with colored rectangles which music fragments correspond with literal transcriptions of the system's output (orange) and to slightly modified transcriptions of variations generated by the tool (green). Unmarked materials were freely written.

A.1 Presentation

Time Pictures is a small piano suite in four movements which was inspired by the physical gestures of four different visual art techniques: oil or acrylic brush strokes, watercolor dissolution, chemical cut in etching and pixel color representations in digital arts. Much of the music present in this work is either a literal transcription of the tool output or a slightly modified version of the system variations. Provided that the system solved many aspects of pitch selection according to my algorithms, much of the composition process was focused in modeling musical gestures in a sort of "diatonic fashion" through improvising them on a MIDI keyboard. Later, different sonorities were "instilled" in each gesture and tested instantly until the more aurally satisfying result was found. Harmonies in fourths and some relevance of the D♯ - E pitch classes are somehow unifying factors through the four movements of the piece.

A.2 Evaluation summary

The tool solved complex pitch selection procedures instantly. It proved specially effective to improvise / create passages that include varied repetitions. The composition process focused on modeling the gestures by playing in the keyboard rather than on a detailed selection of pitches and dispositions. Sensorial evaluation prevailed as the main criterion for selecting the music materials that were finally included in the piece.

There were also some desirable improvements that can be done in future work, such as the possibility of changing the pitch class set in the middle of a variation playback, changing the variations playback tempo and the possibility of using open dispositions or perform range stretching operations with the variants. The following pages include the two versions of the music score as explained in the beginning.

Alonso Torres
(San José, 1980)

Time Pictures

for piano

to Lilly

I. Brush Strokes and Spots
II. Watercolors
III. Etching
IV. Pixels

Duration: approx. 7':05"

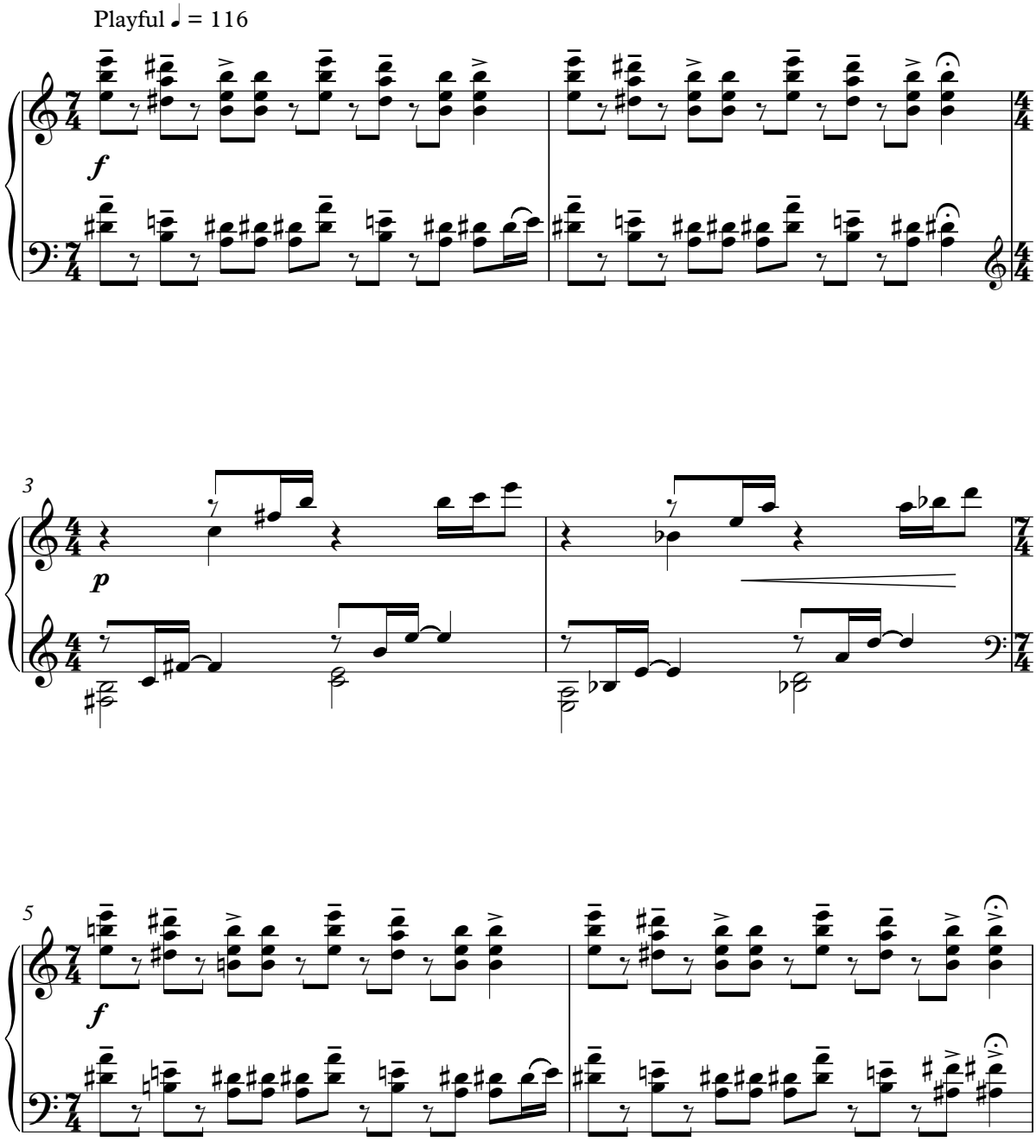
Time Pictures

for Piano

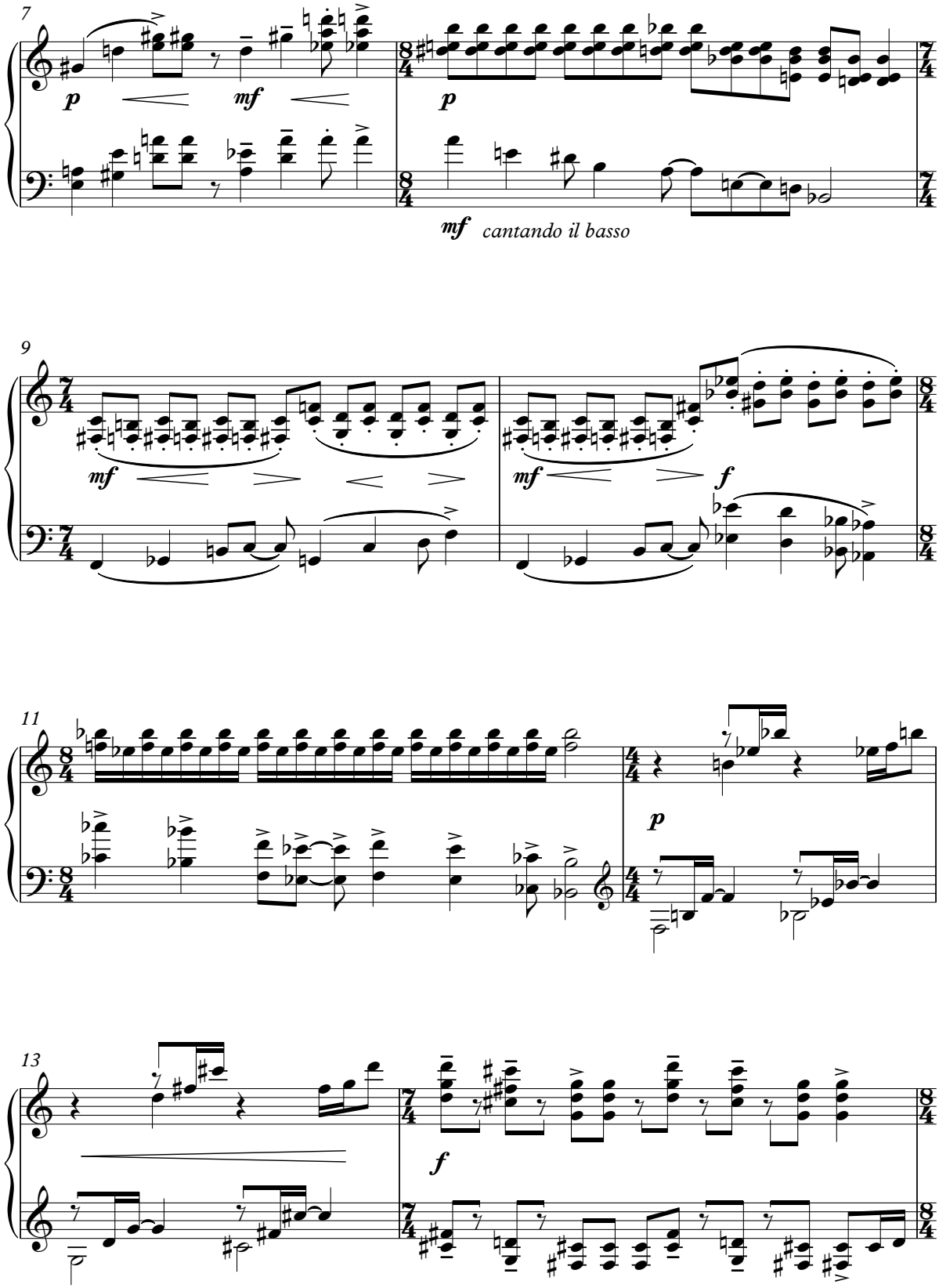
to Lilly

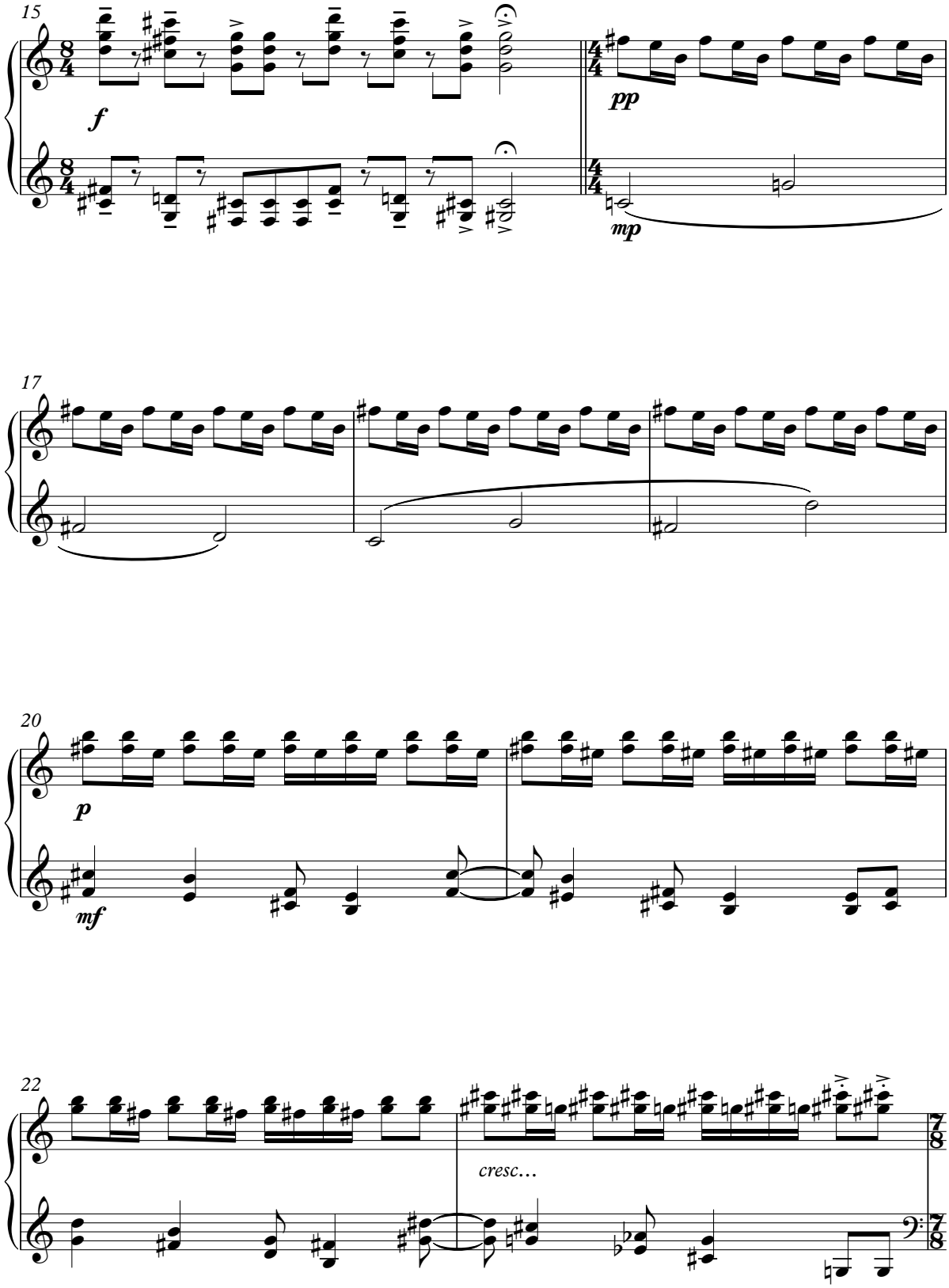
I. Brush strokes and spots

Alonso Torres

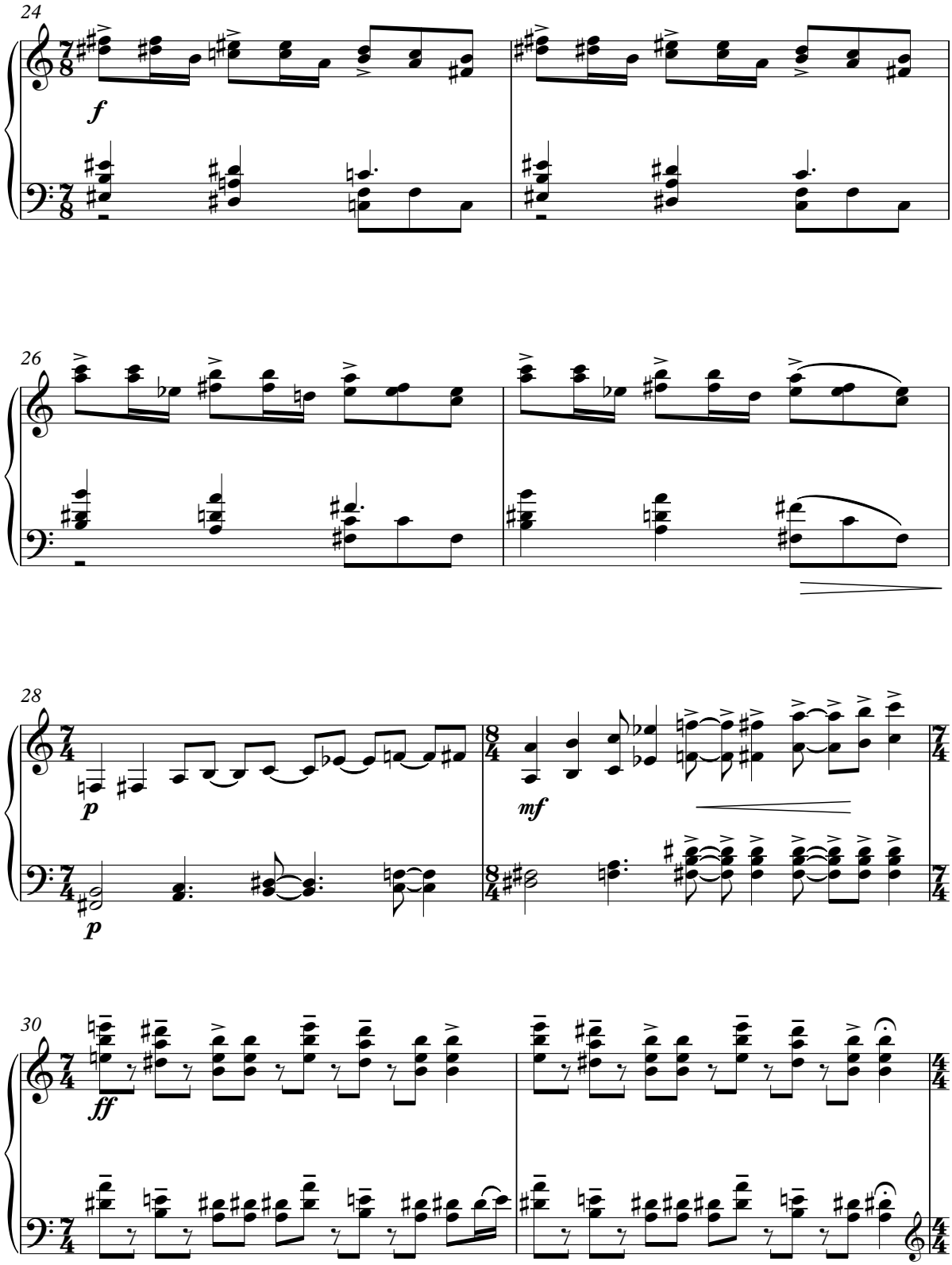


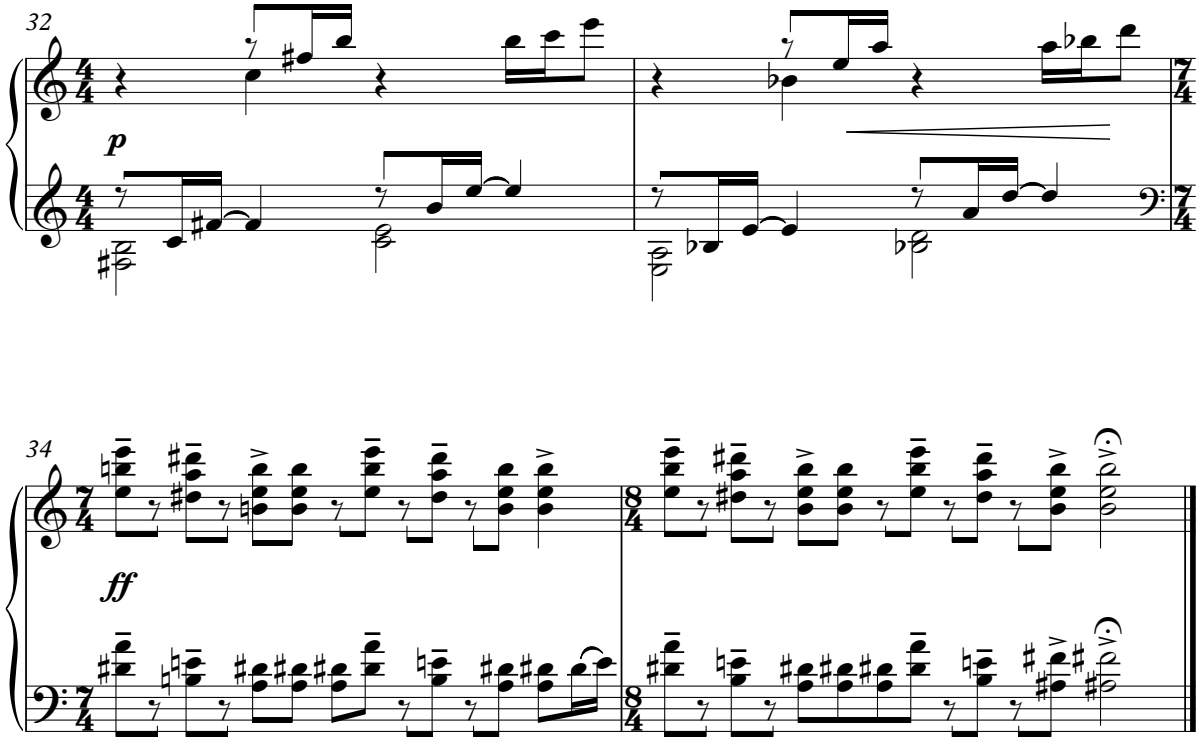
Alonso Torres - Time Pictures



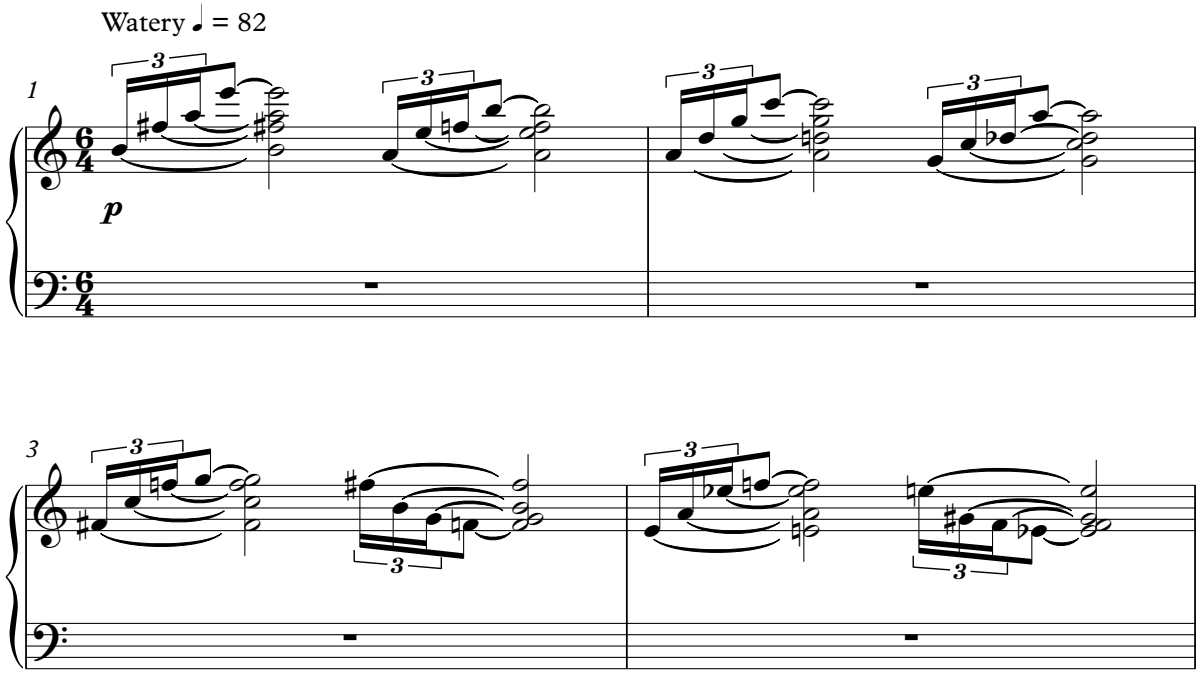

15
f
pp
mp
17
20
p
mf
22
cresc…

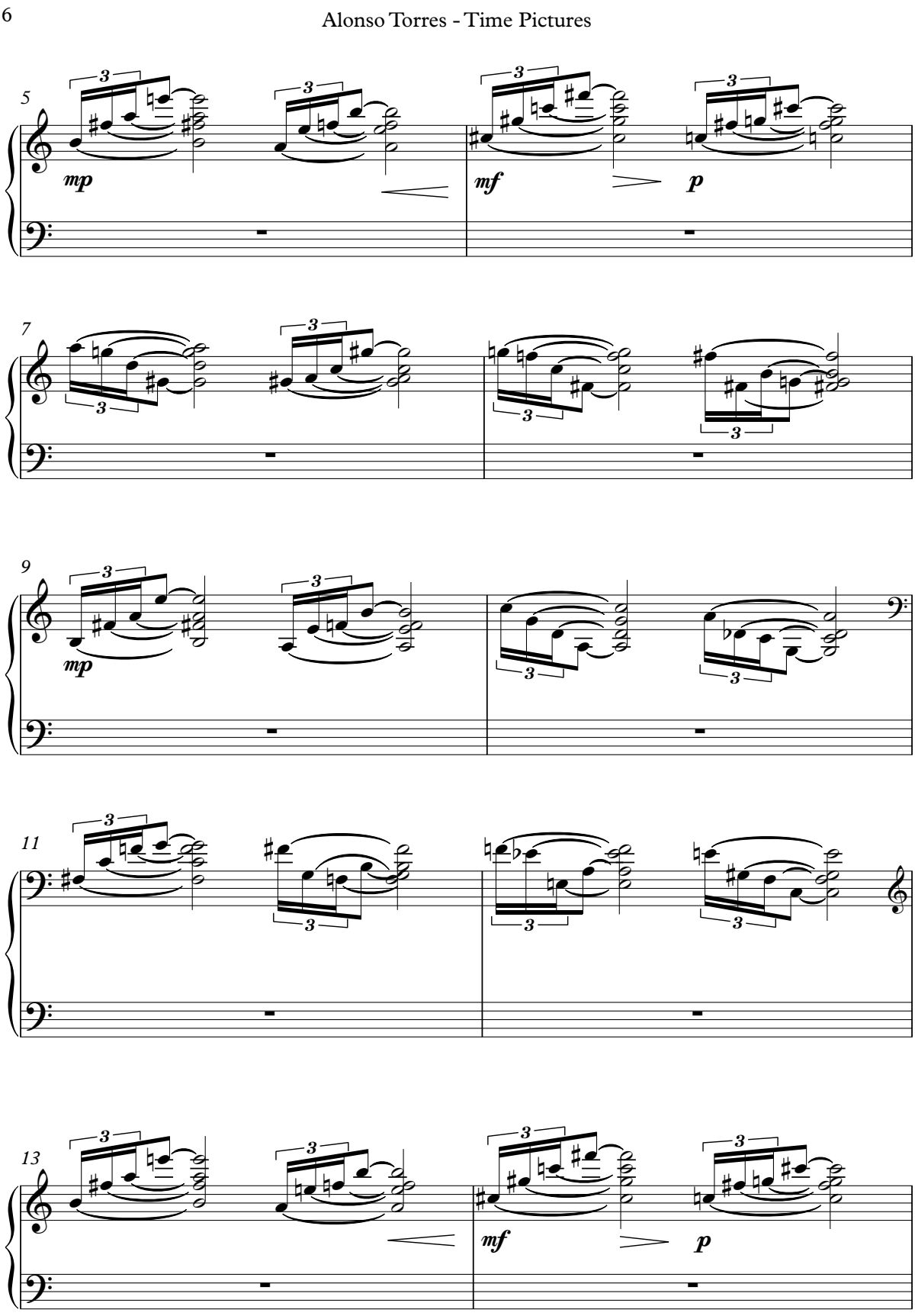
Alonso Torres - Time Pictures

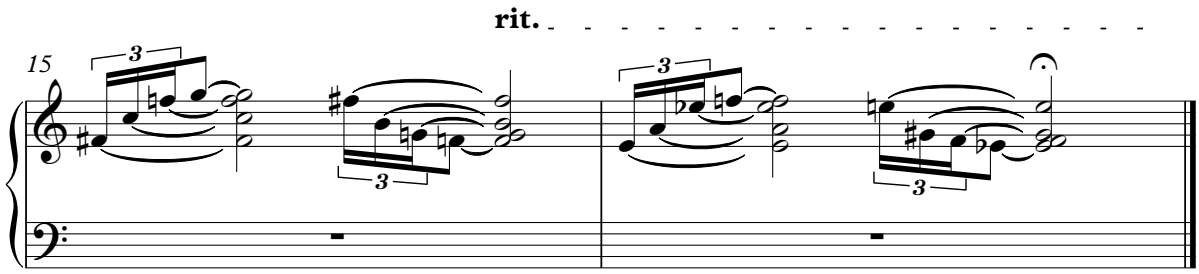




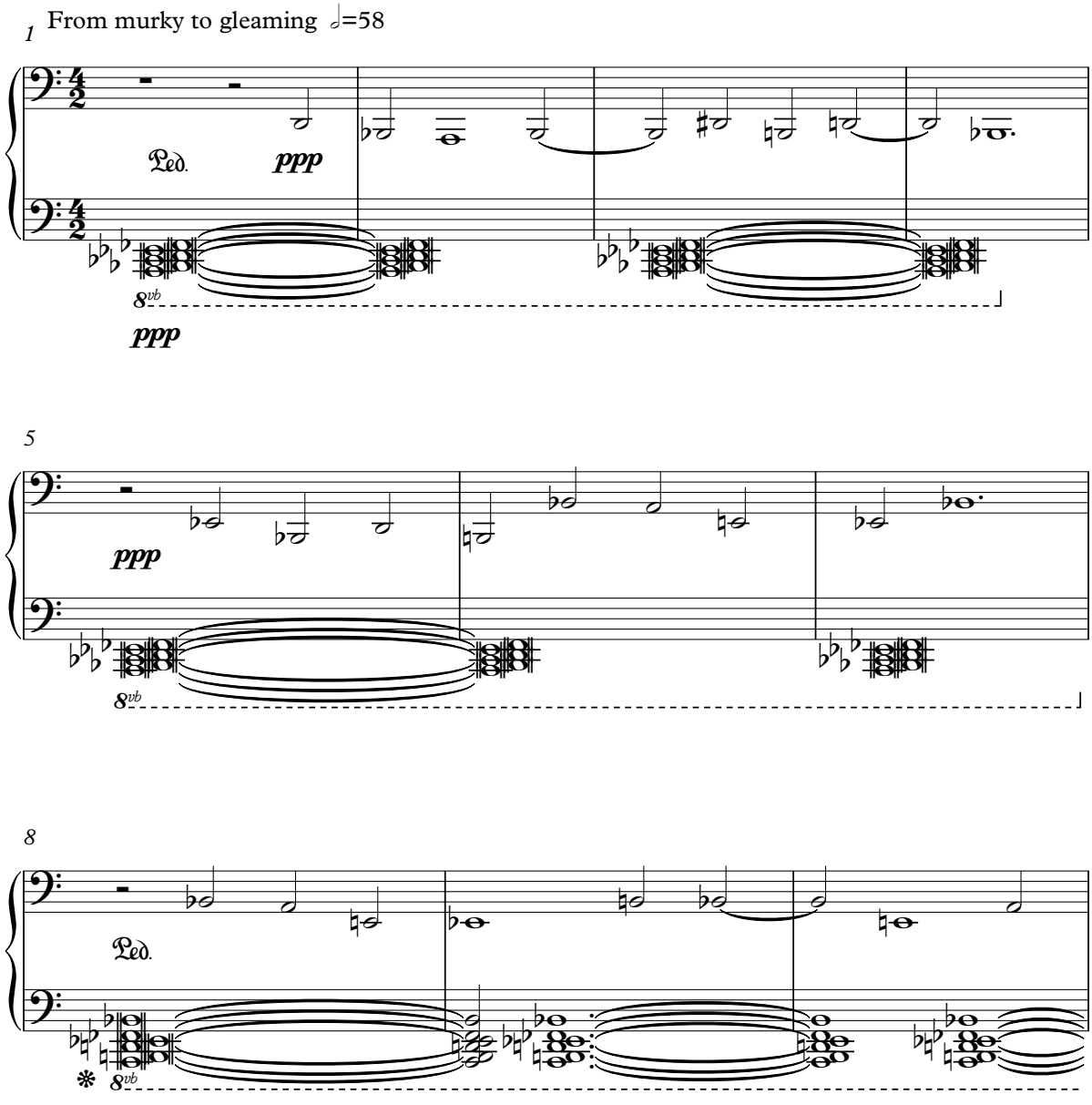
II. Watercolors




6
Alonso Torres - Time Pictures
5
mp
mf
p
7
9
mp
11
13
mf
p

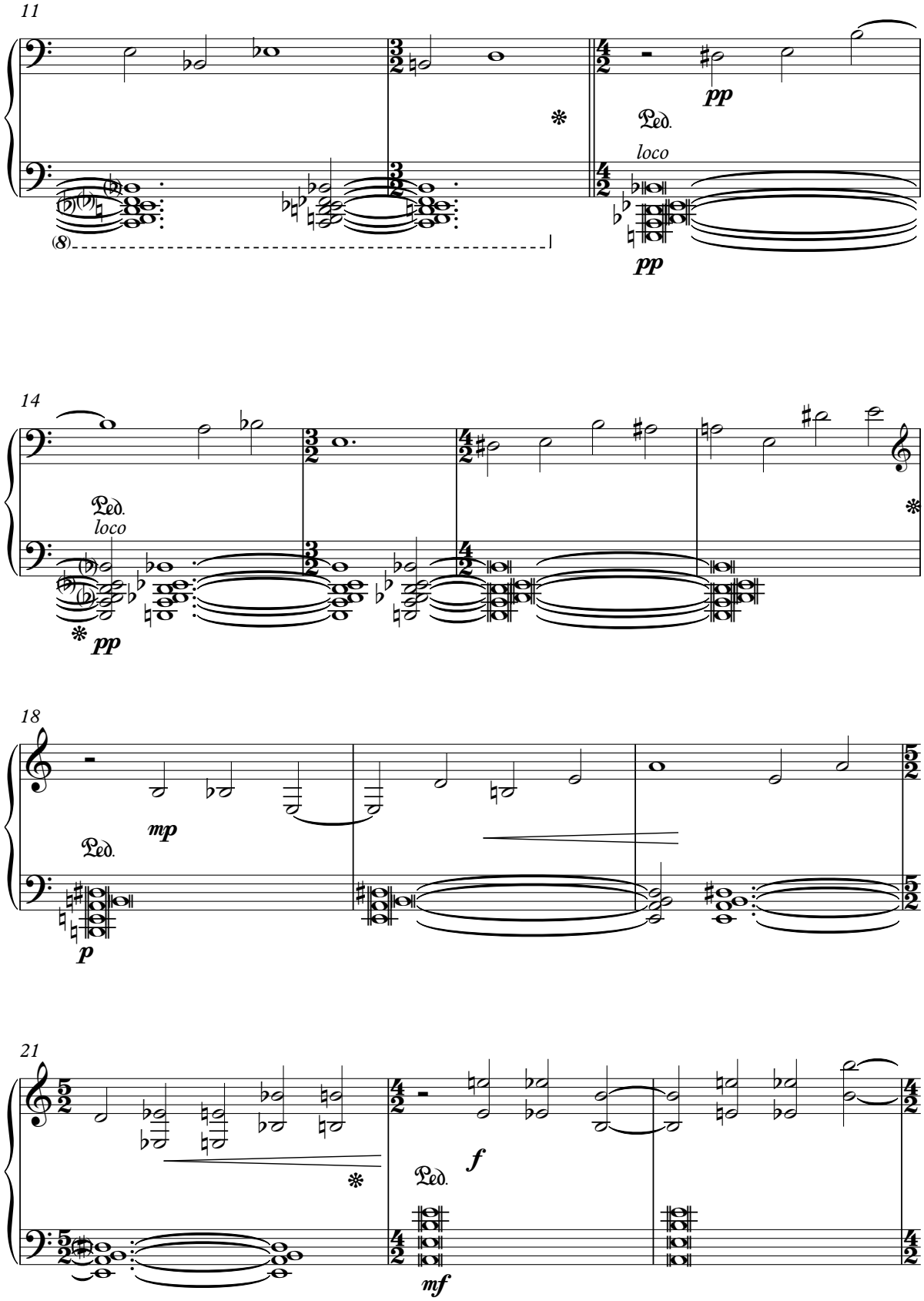

rit.
15
3
3
3
3

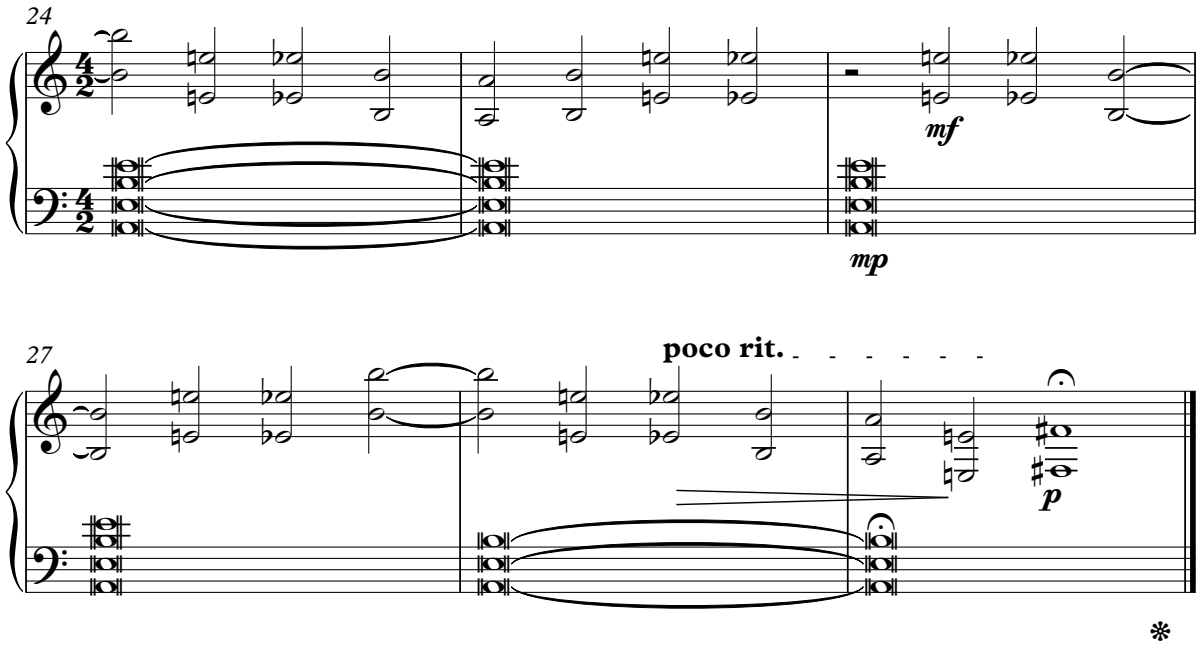
III. Etching


1
From murky to gleaming ♩=58
Ped.
ppp
8vb
ppp
5
ppp
8vb
8
Ped.
8vb

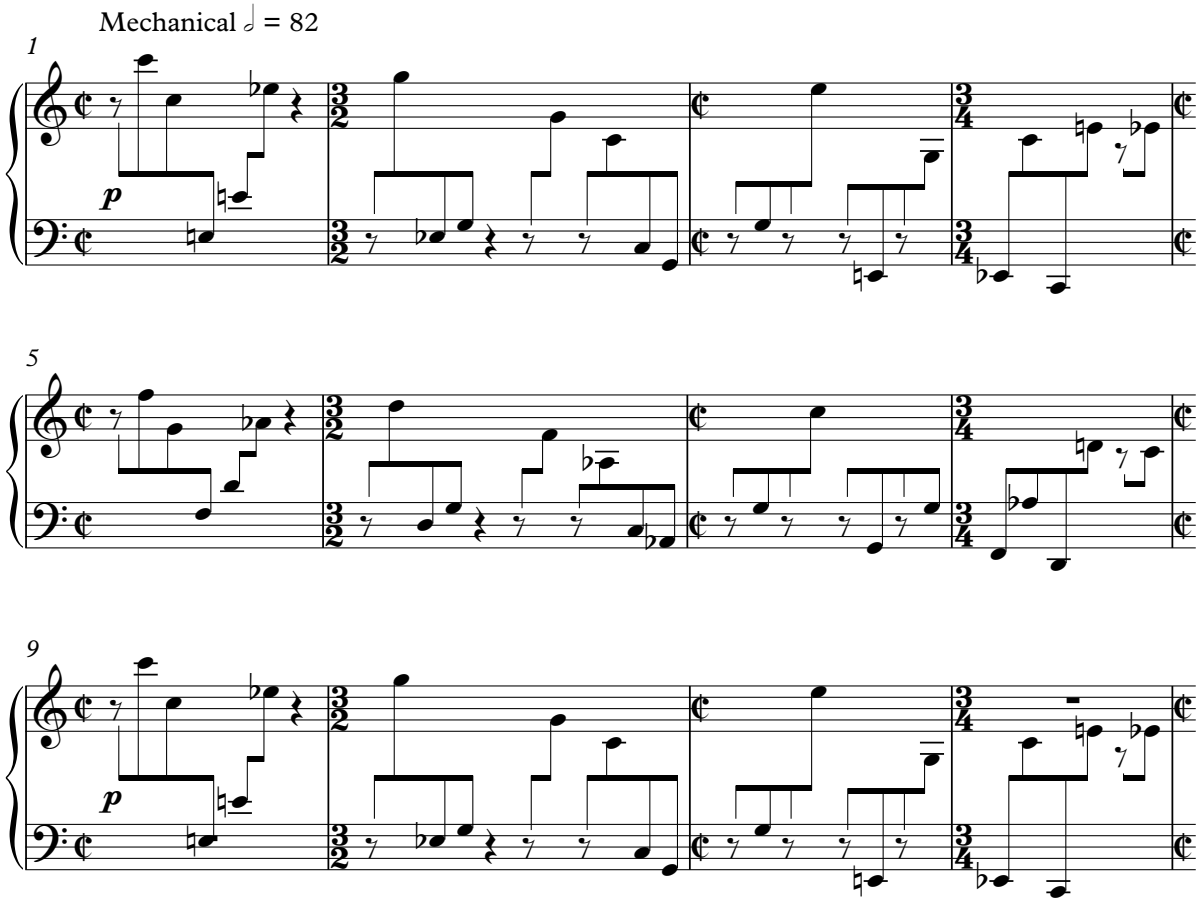
8

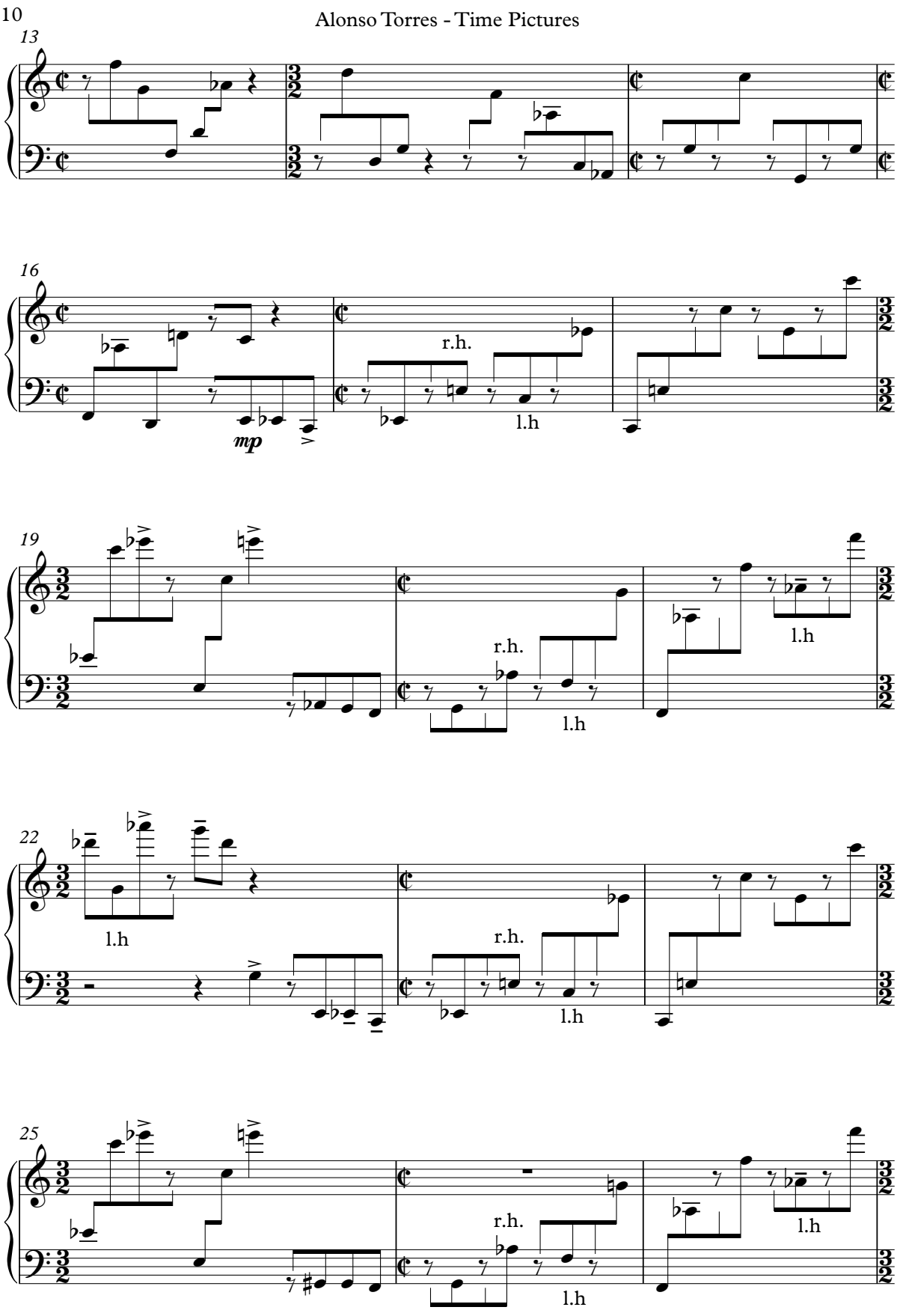
Alonso Torres - Time Pictures

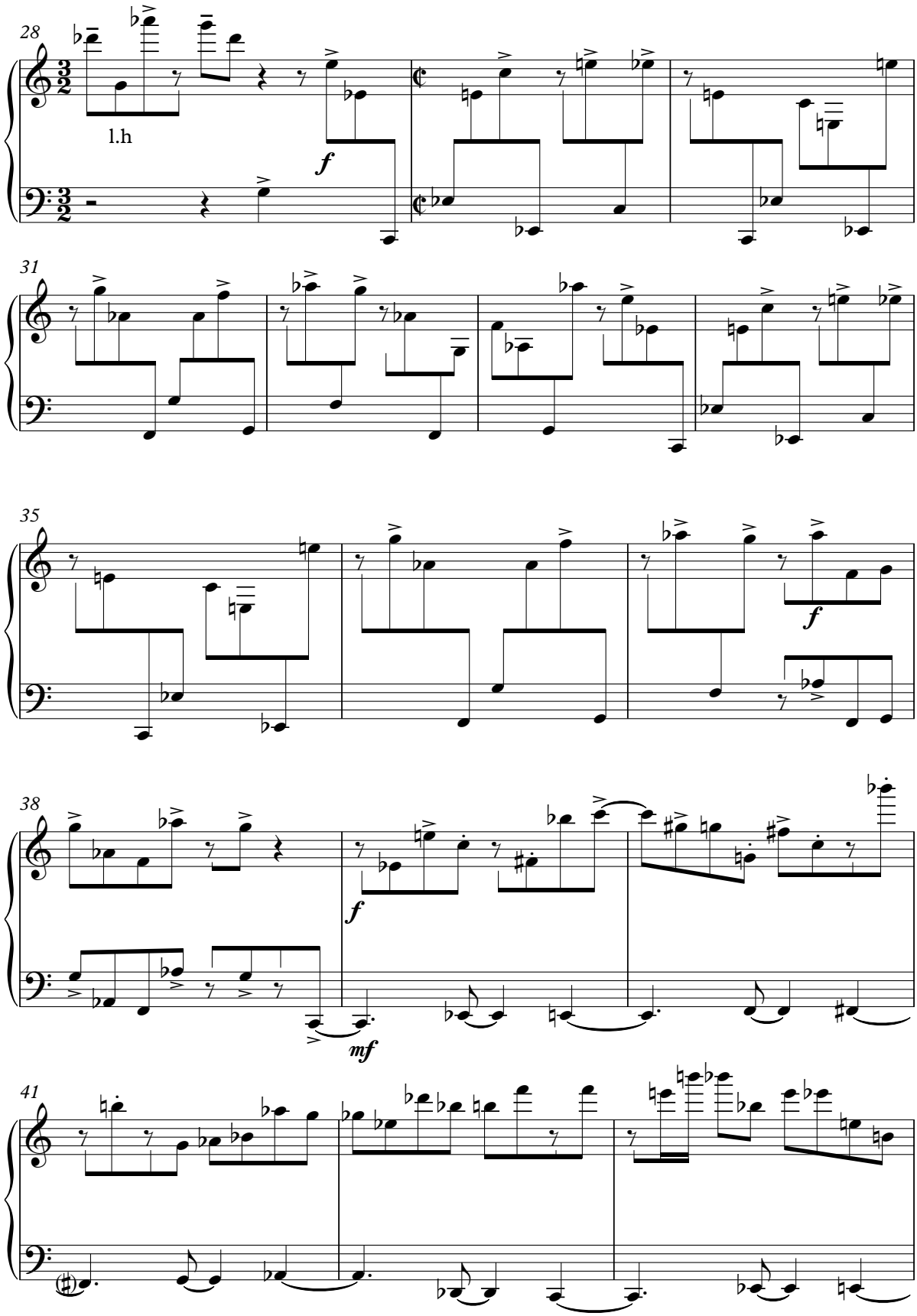


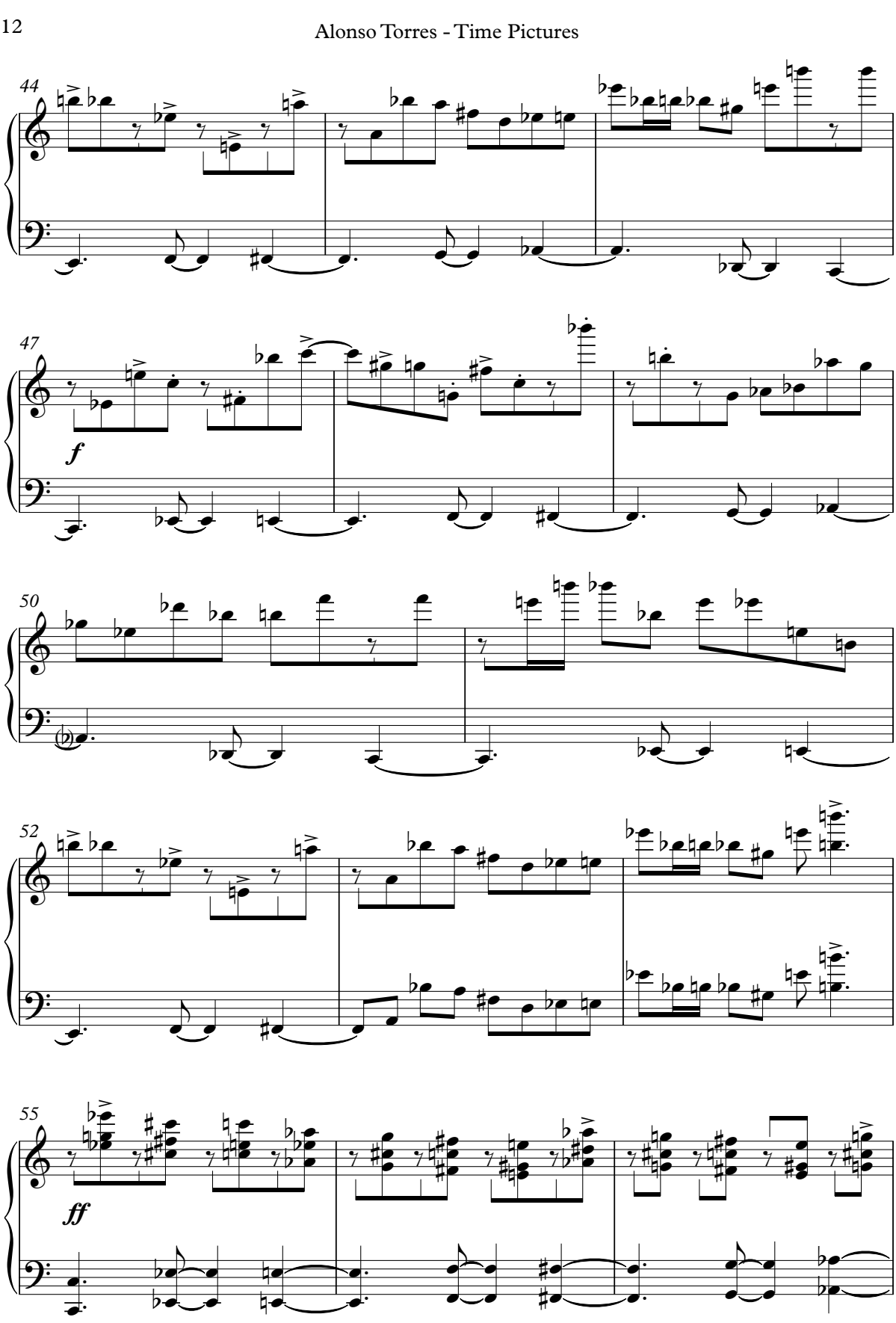


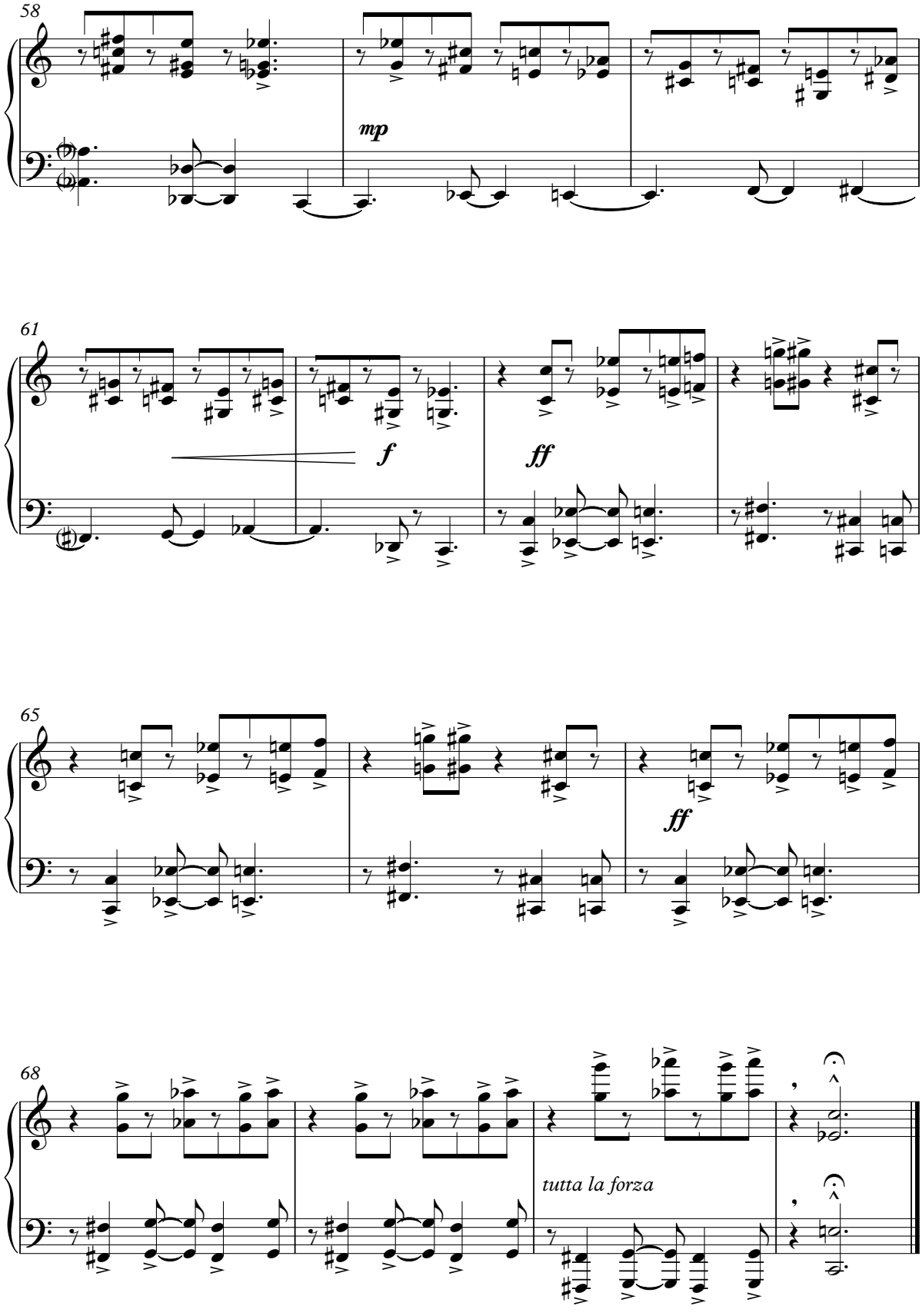
IV. Pixels




10
Alonso Torres - Time Pictures
13
16
mp
r.h.
l.h
19
r.h.
l.h
22
l.h
r.h.
l.h
25
r.h.
l.h


28
l.h
f
31
35
f
38
f
mf
41


12
Alonso Torres - Time Pictures
44
47
f
50
52
55
ff



Porto, julio 2018

Time Pictures

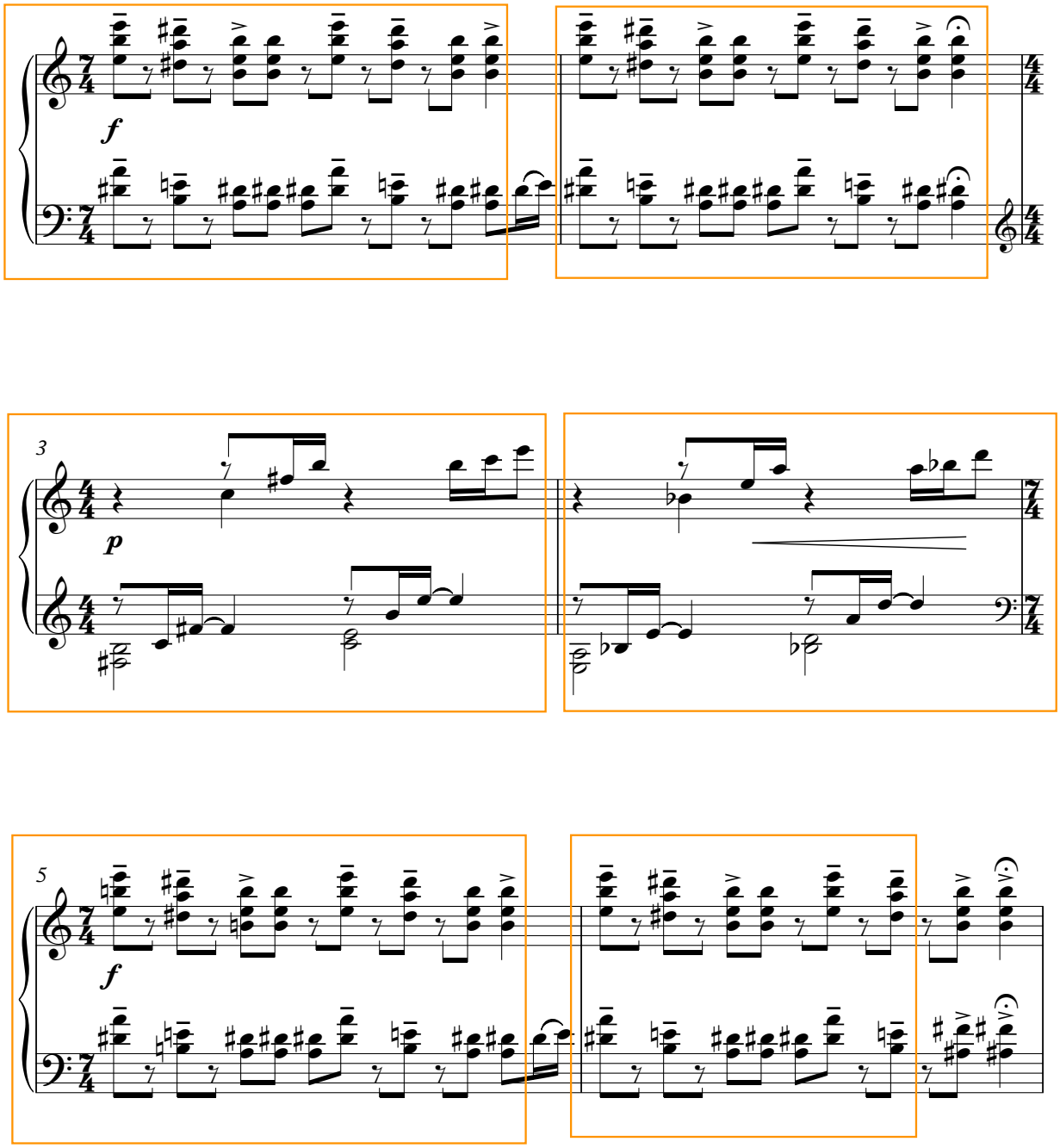
for Piano

to Lilly

I. Brush strokes and spots

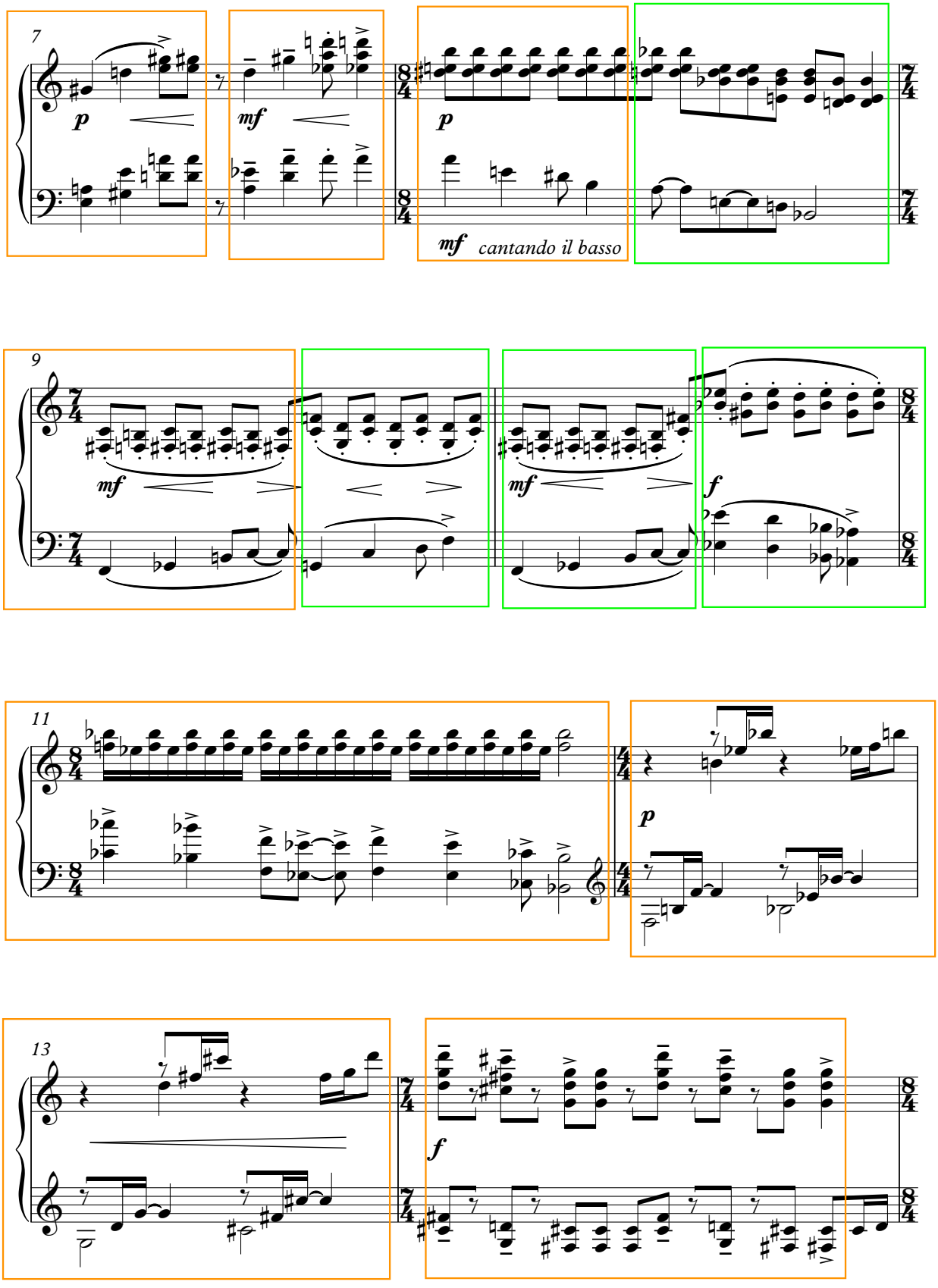
Alonso Torres

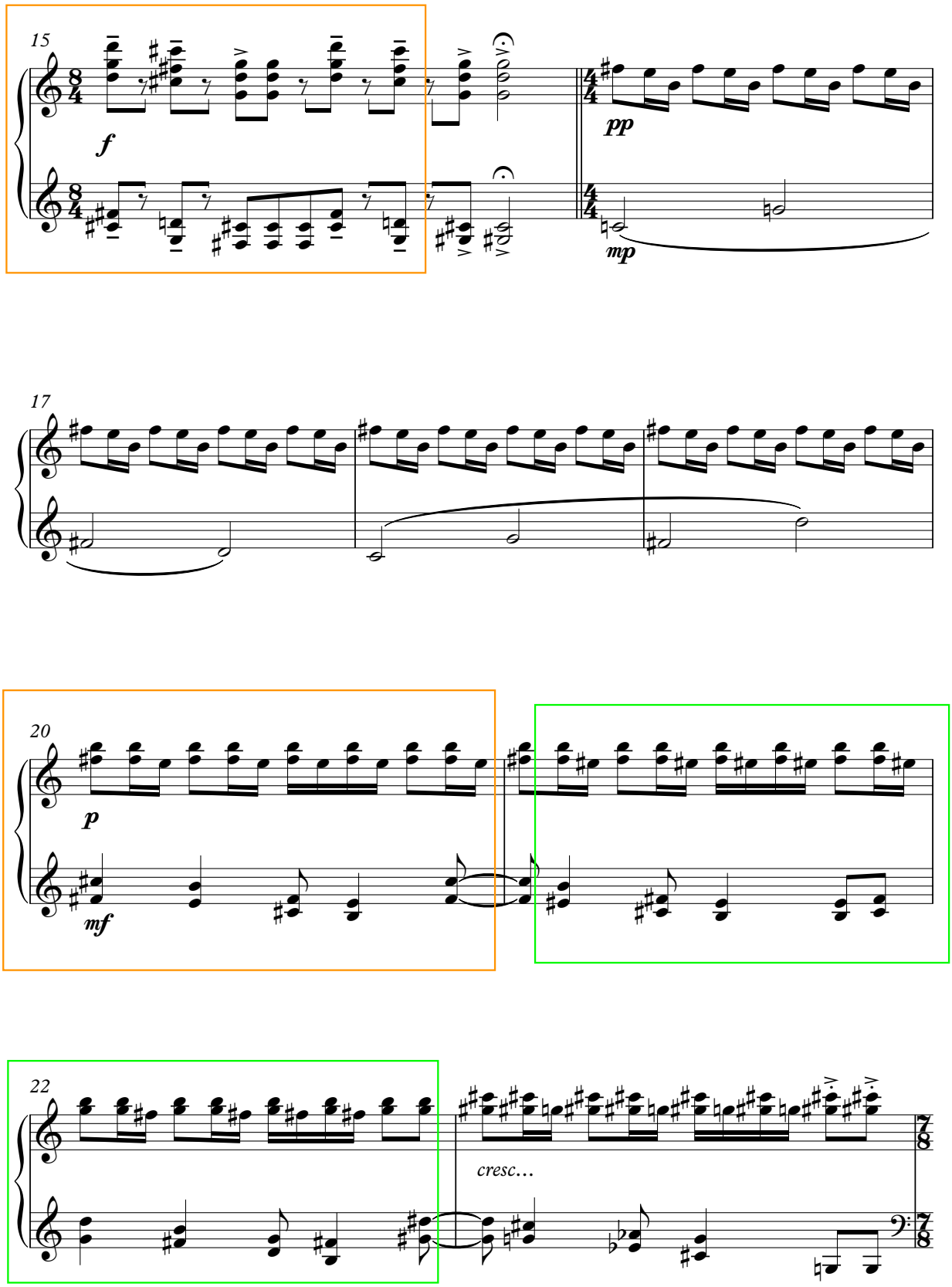
Playful ♩ = 116



2

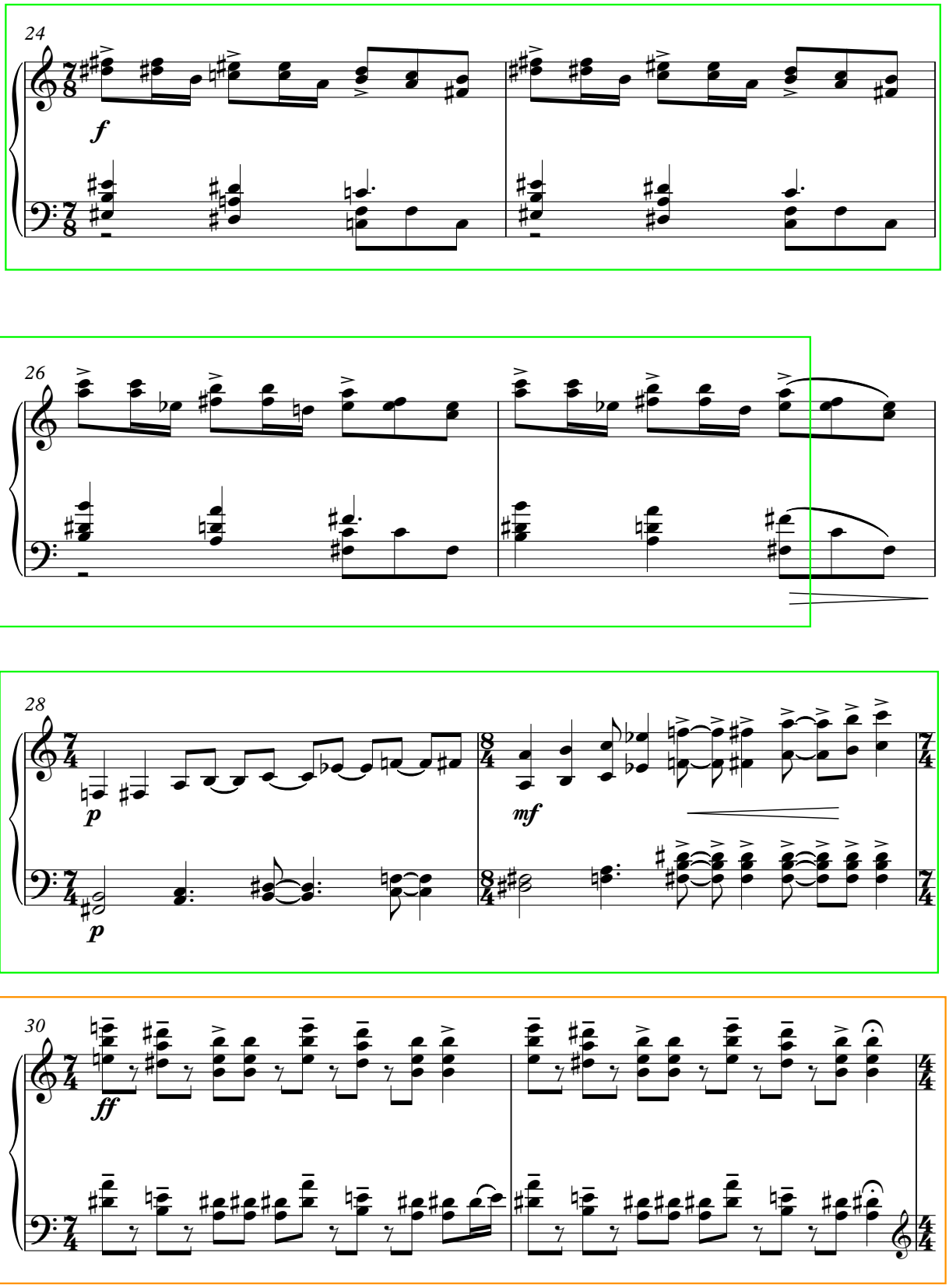
Alonso Torres - Time Pictures

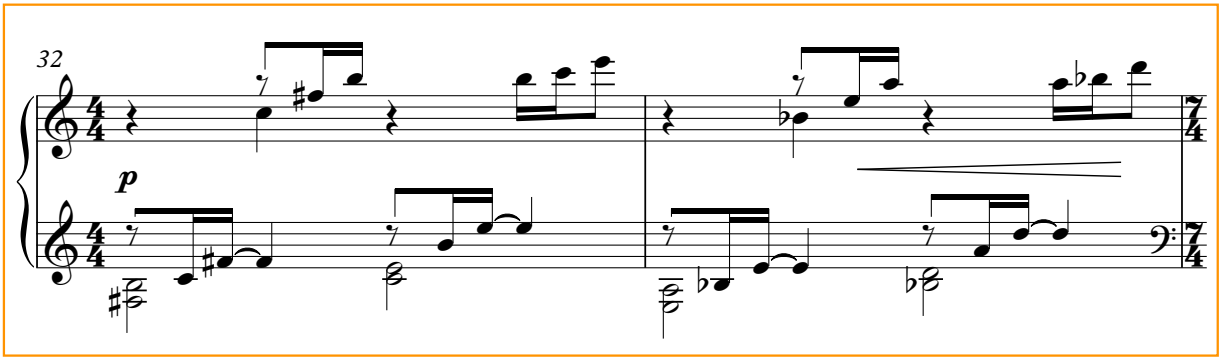


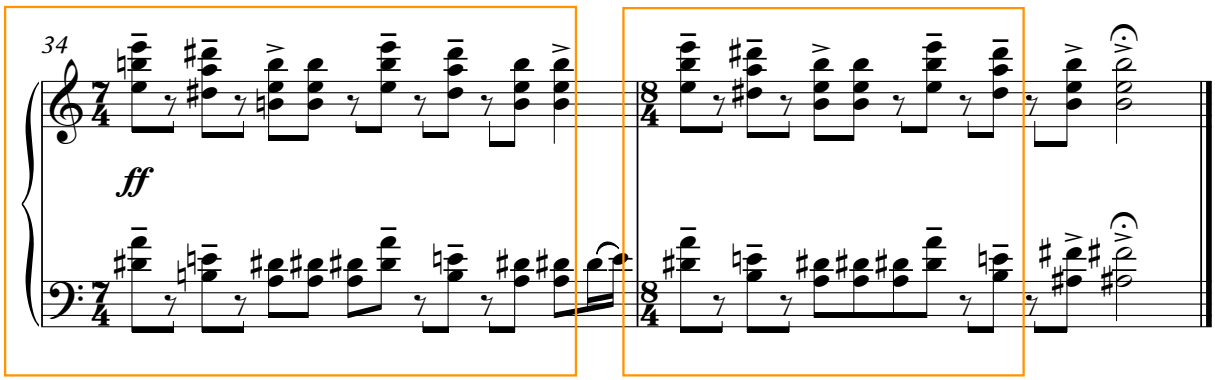

15
f
pp
mp
17
20
p
mf
22
cresc…

4

Alonso Torres - Time Pictures

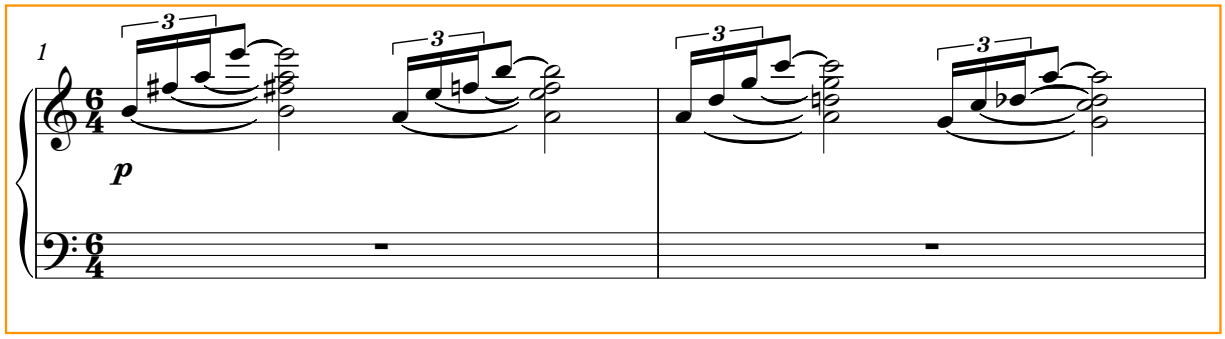


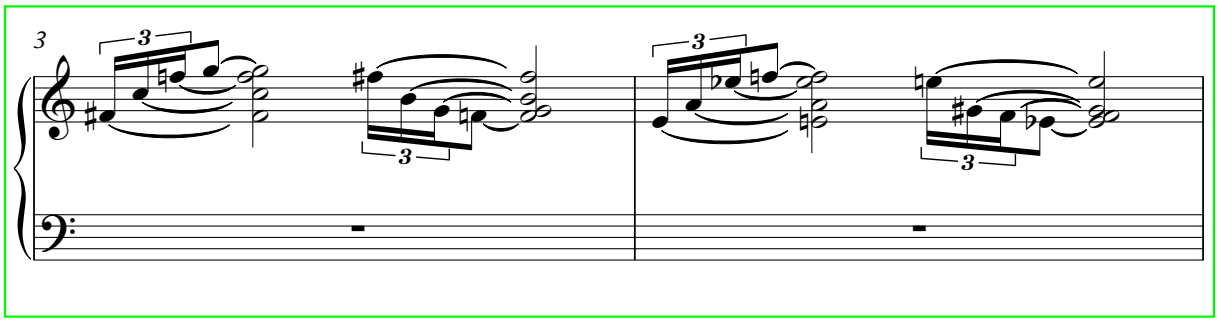




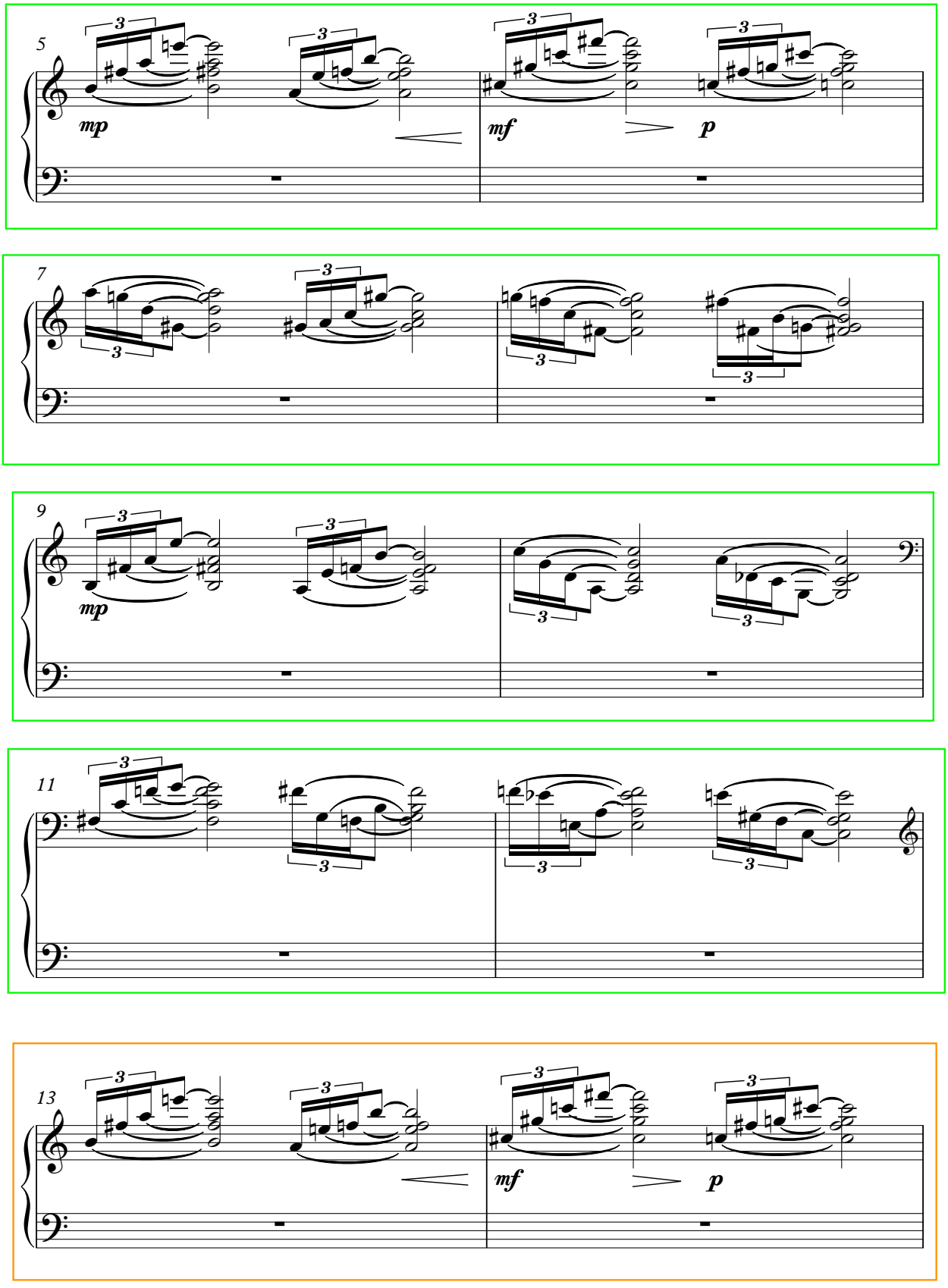
II. Watercolors

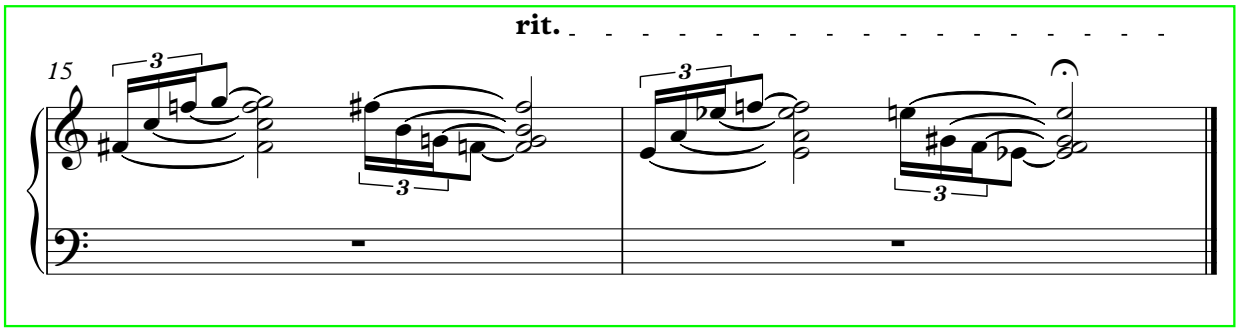
Watery ♩ = 82



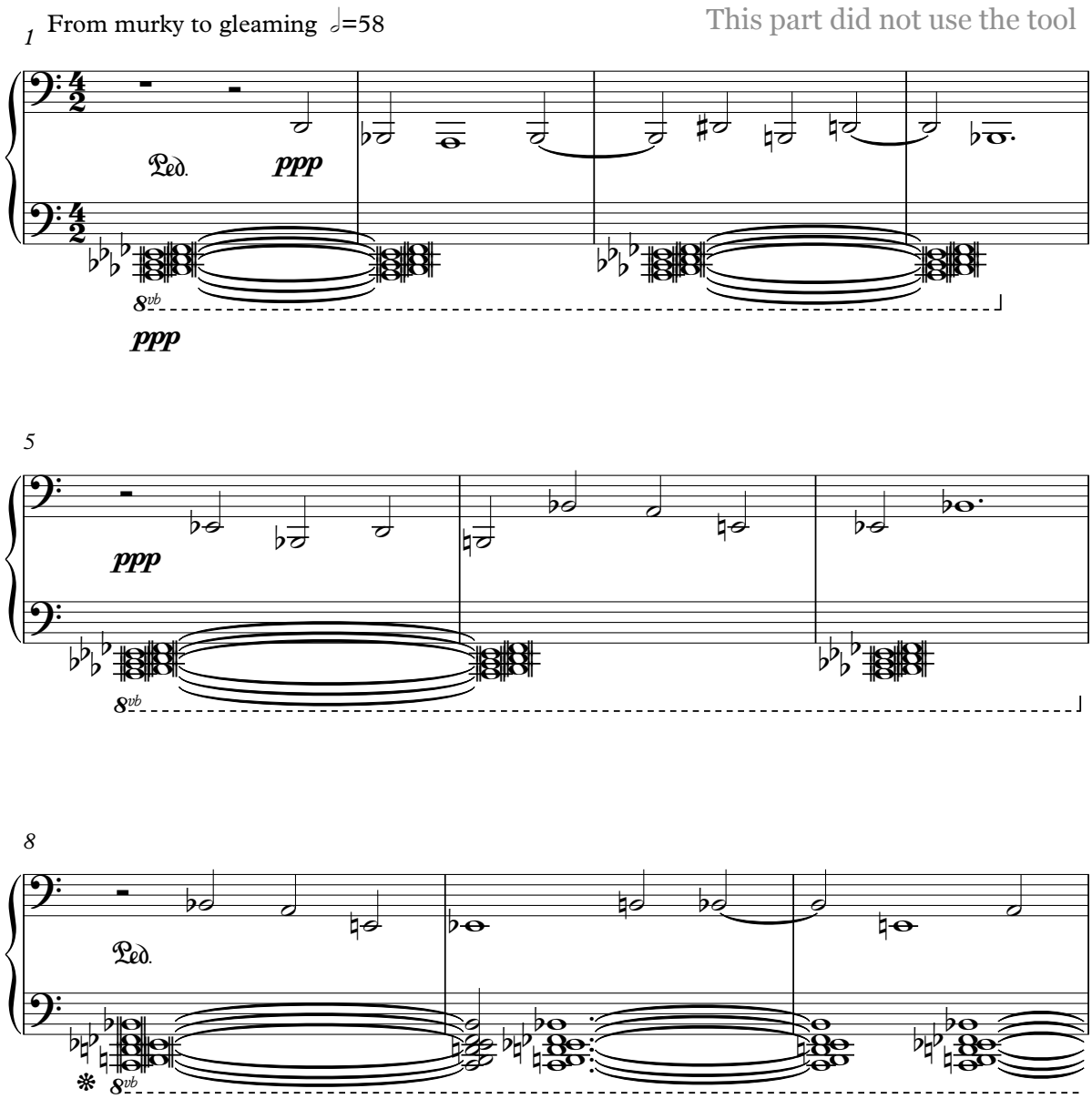


Alonso Torres - Time Pictures

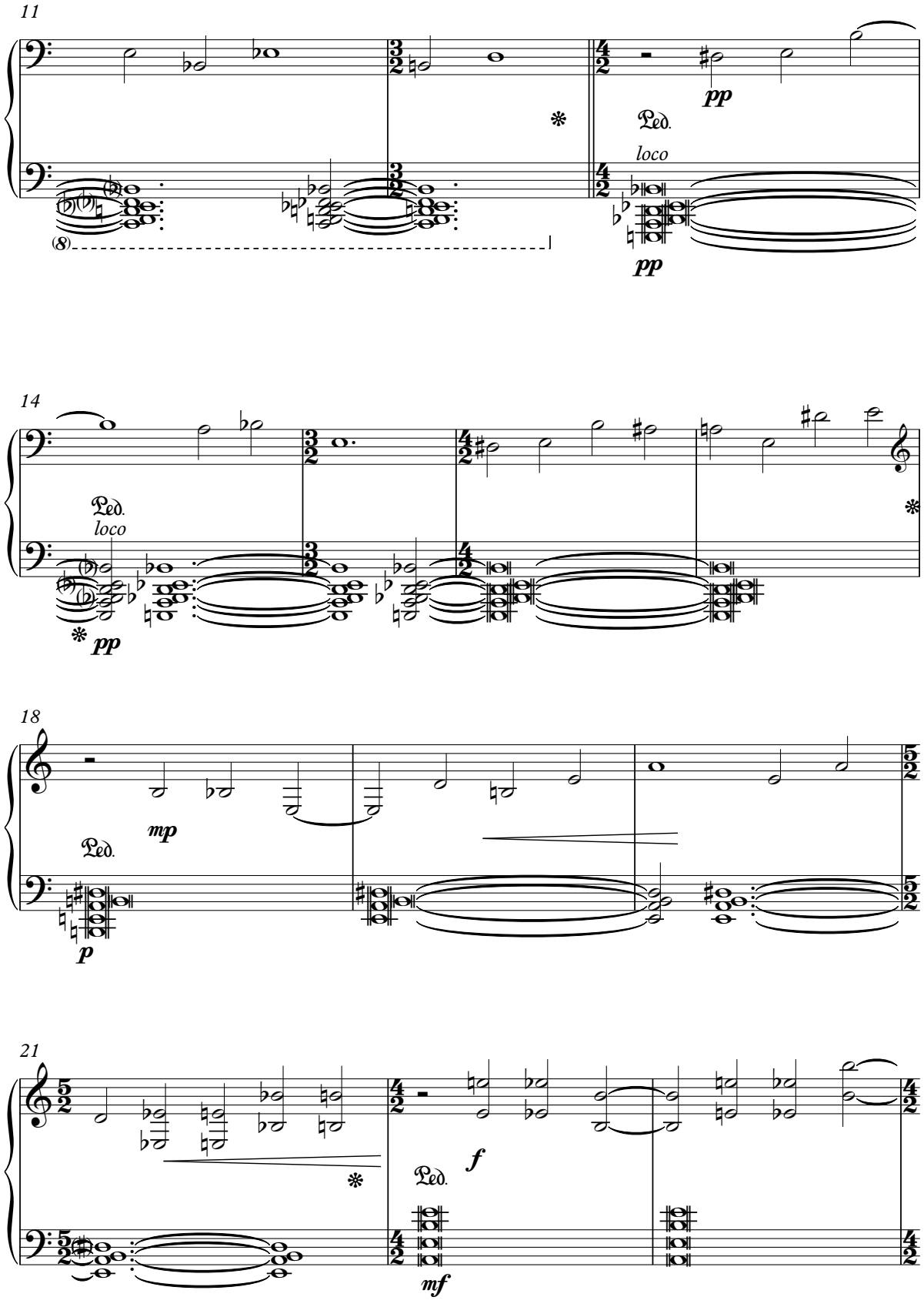


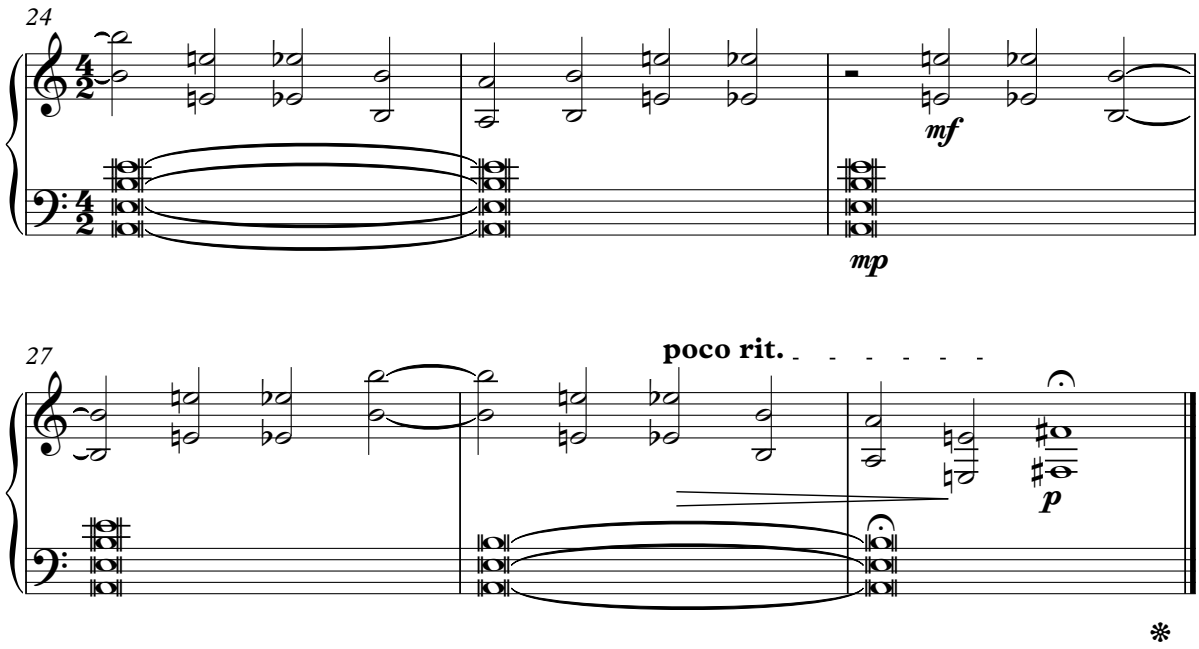

rit.
15
3
3
3
3

III. Etching

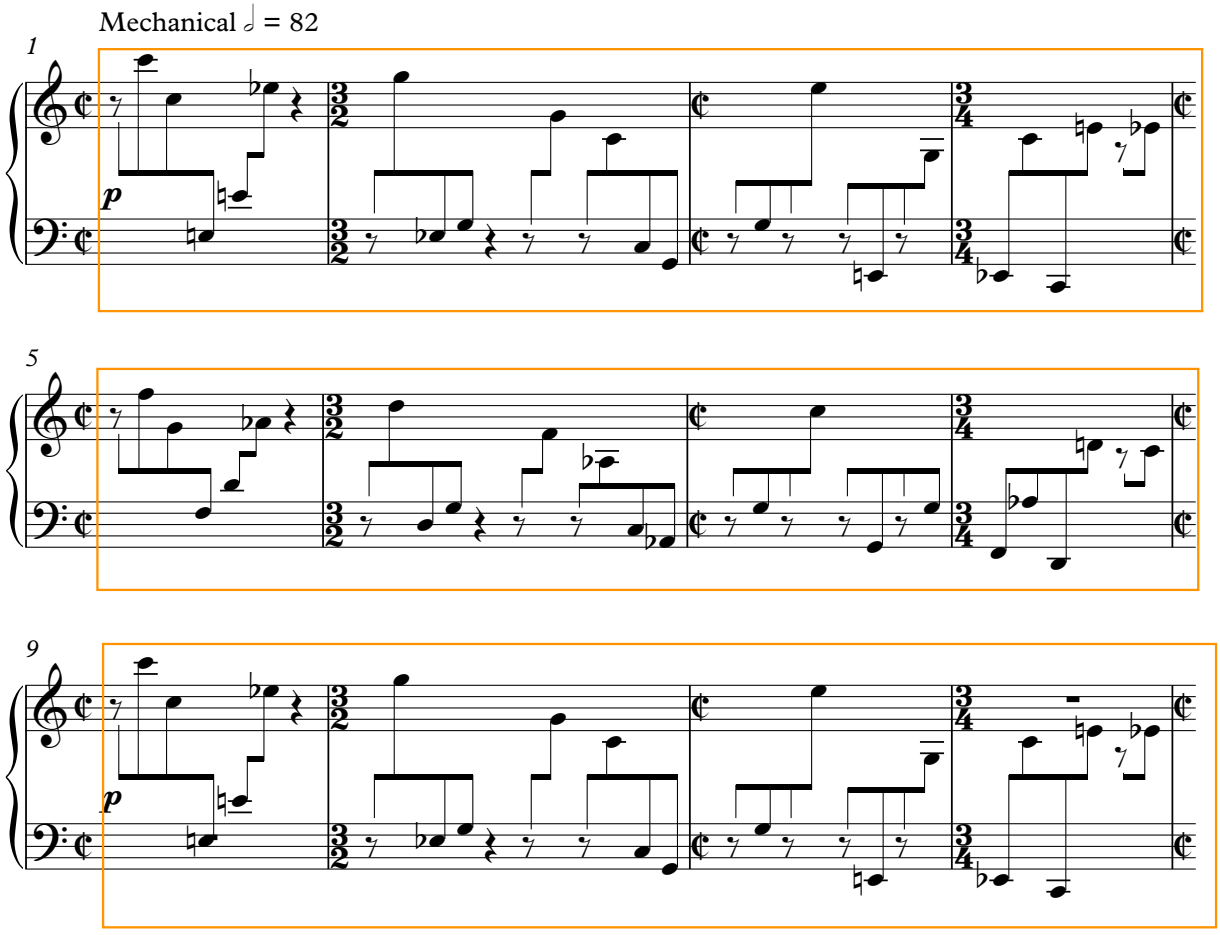

1
From murky to gleaming ♩=58
This part did not use the tool
Ped.
ppp
8vb
ppp
5
ppp
8vb
8
Ped.
8vb

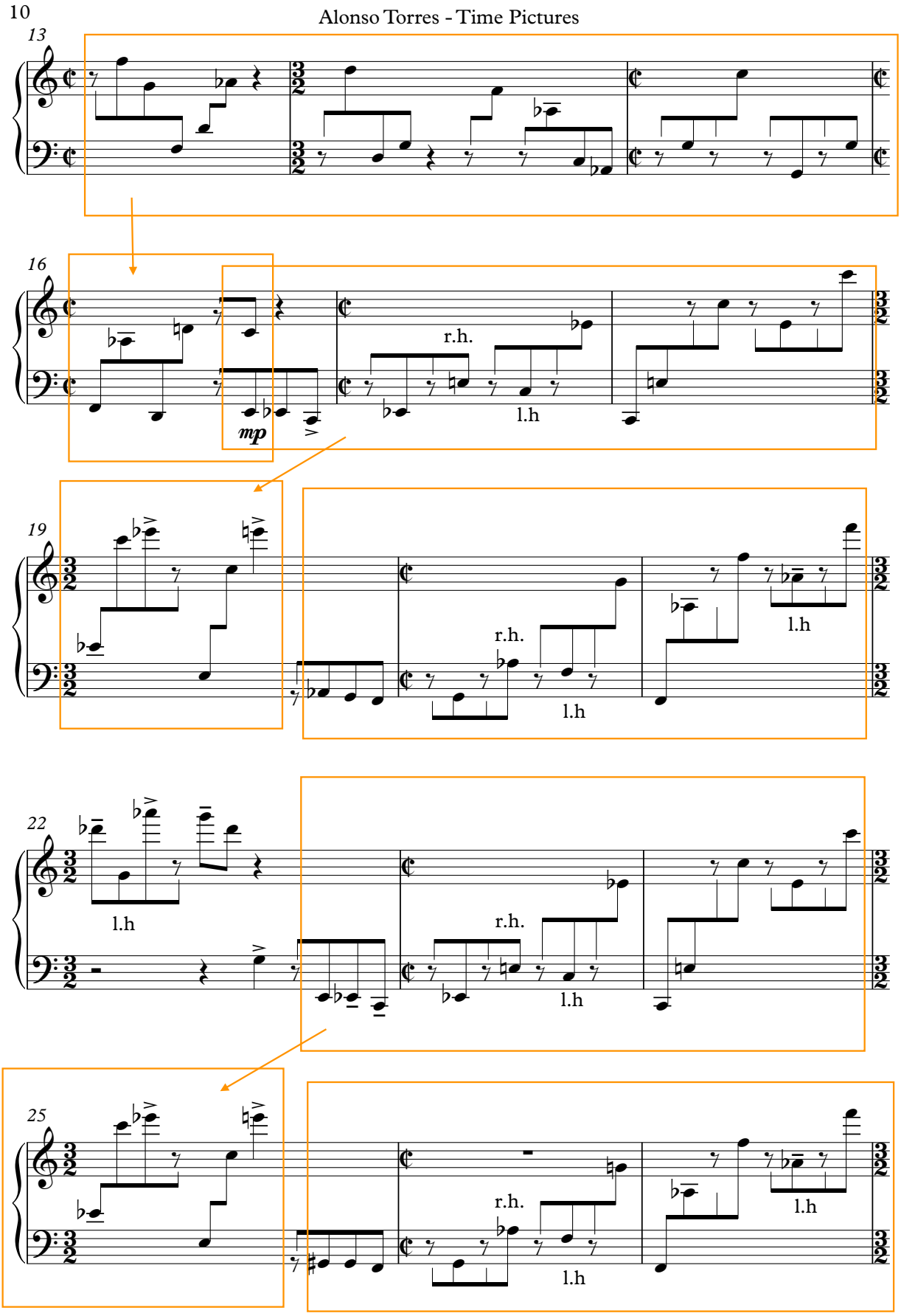
Alonso Torres - Time Pictures

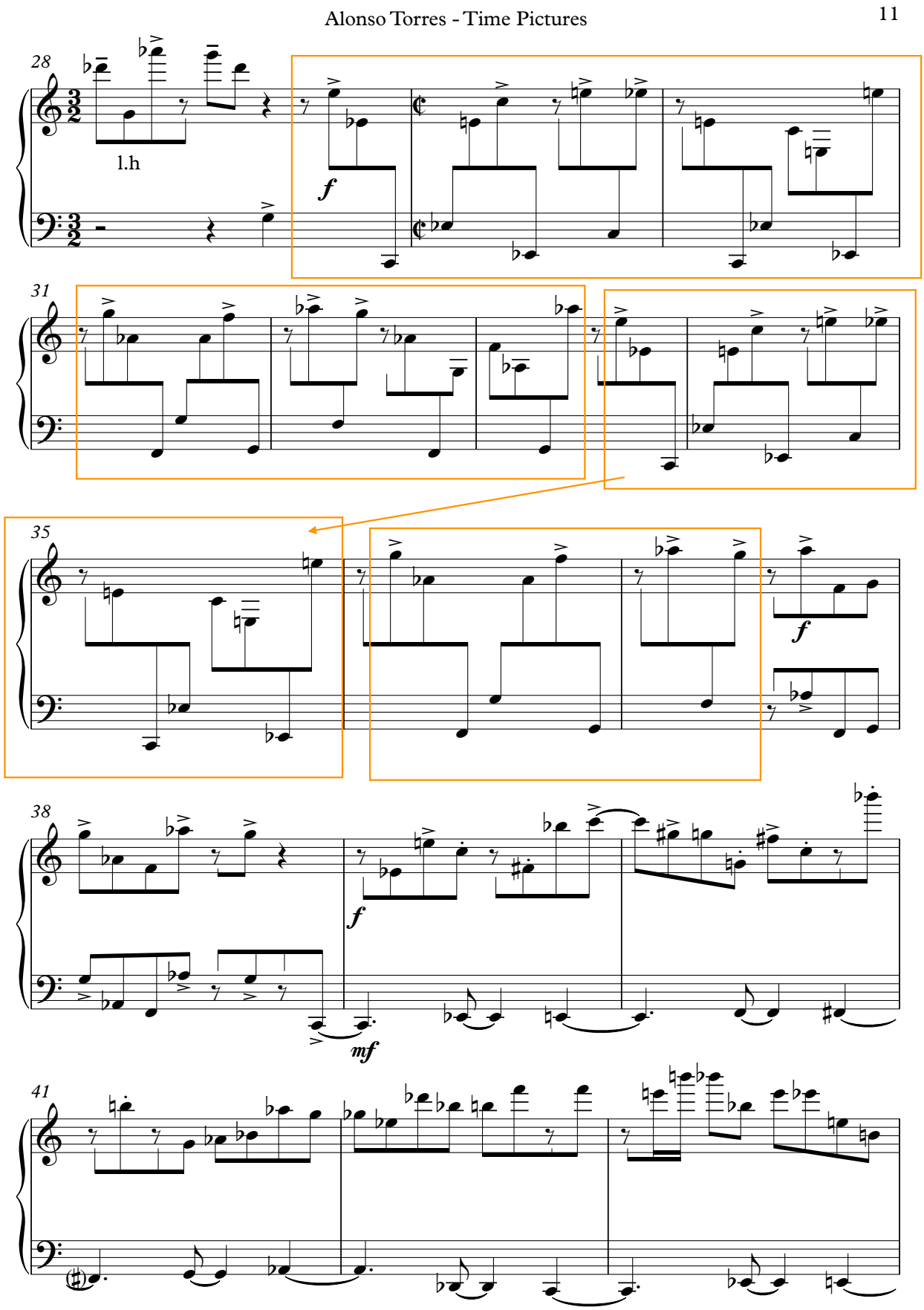




IV. Pixels

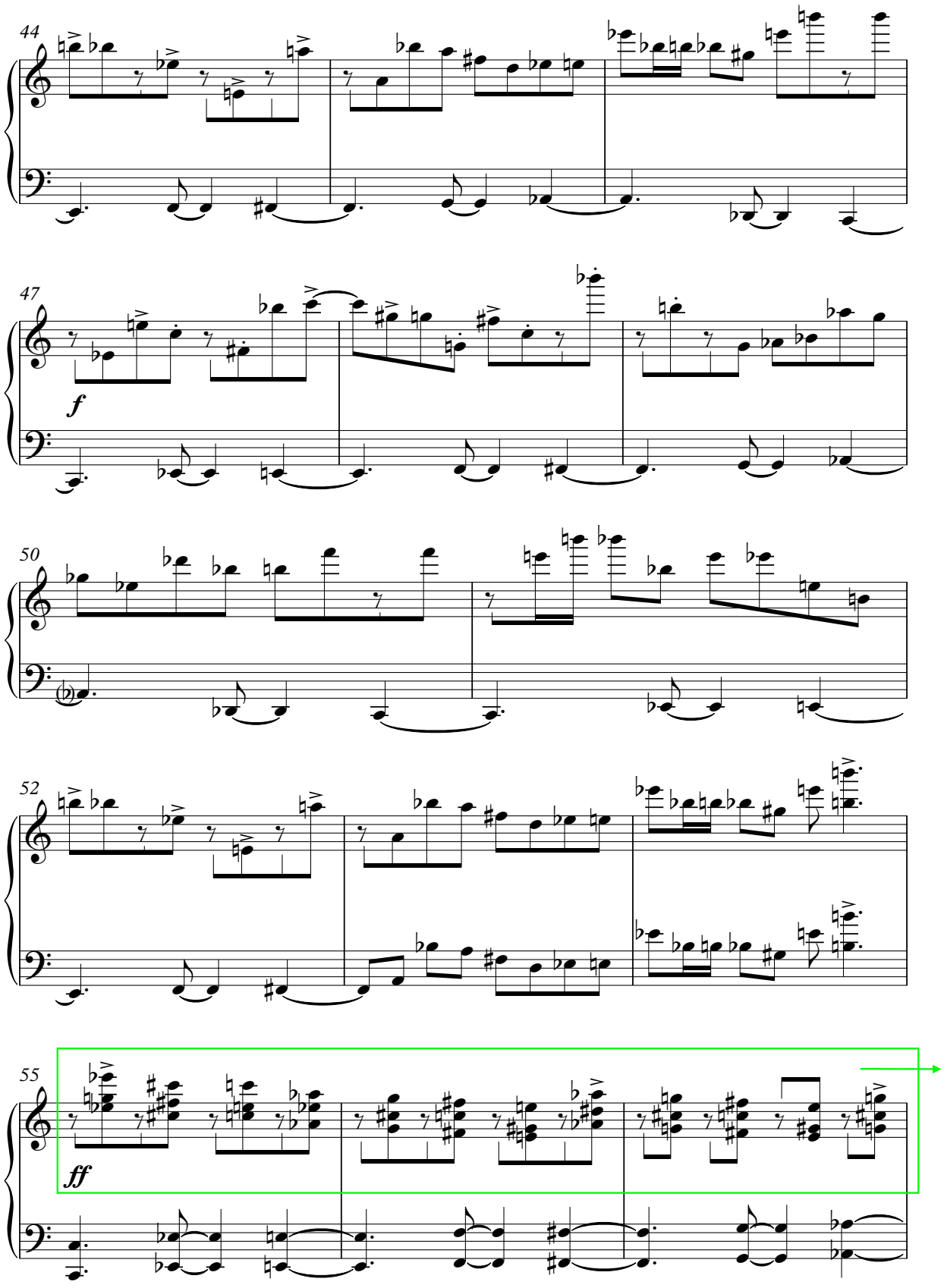



10
Alonso Torres - Time Pictures
13
16
mp
r.h.
l.h
19
r.h.
l.h
l.h
22
l.h
r.h.
l.h
25
r.h.
l.h
l.h

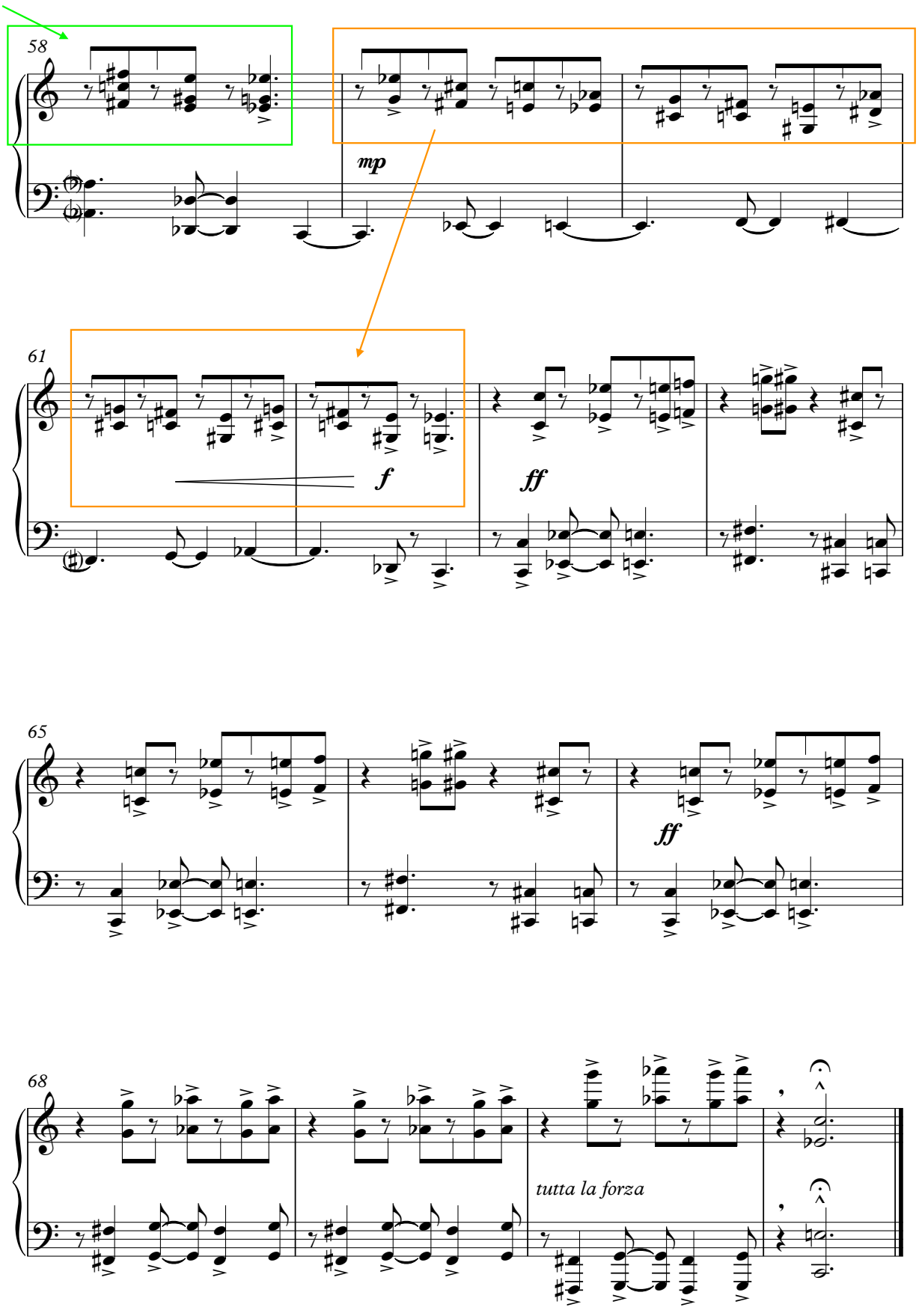

Alonso Torres - Time Pictures
11
28
l.h
f
31
35
f
38
f
mf
41

12

Alonso Torres - Time Pictures



Alonso Torres - Time Pictures 13



Porto, julio 2018

Bibliography

[Agostini and Ghisi, 2012] Agostini, A. and Ghisi, D. (2012). bach: an environment for computer-aided composition in max. In *Proceedings of the International Computer Music Conference (ICMC 2012)*, pages 373–378, Ljubljana, Slovenia.

[Archer, 1995] Archer, B. (1995). The nature of research. *Co-design, interdisciplinary journal of design*, pages 6–13.

[Ariza, 2005] Ariza, C. (2005). Navigating the landscape of computer aided algorithmic composition systems: a definition, seven descriptors, and a lexicon of systems and research. In *Proceedings of the International Computer Music Conference (ICMC 2005)*, Barcelona, Spain.

[Assayag, 1998] Assayag, G. (1998). Computer assisted composition today. In *First Symposium on music and computers.*, pages 23–25, Corfu.

[Assayag et al., 1999] Assayag, G., Rueda, C., Laurson, M., Agon, C., and Delerue, O. (1999). Computer-assisted composition at ircam: From patchwork to openmusic. *Computer Music Journal*, 23(3):59–72.

[Belkin, 2018] Belkin, A. (Consulted in january 2018). General principles of harmony.

[Bor, 2009] Bor, M. (2009). *Contour reduction algorithms: a theory of pitch and duration hierarchies for post-tonal music*. PhD thesis, University of British Columbia.

[Bresson et al., 2017] Bresson, J., Bouche, D., Carpentier, T., Schwarz, D., and Garcia, J. (2017). Next-generation computer-aided composition environment: A new implementation of open music. In *Proceedings of the International Computer Music Conference (ICMC 2017)*, Shangai, China.

[Cuthbert and Ariza, 2010] Cuthbert, M. and Ariza, C. (2010). music21: A toolkit for computer-aided musicology and symbolic music data. In *11th International Society for Music Information Retrieval Conference*, pages 637–642, Utrecht, Netherlands.

[Cycling'74, 2018] Cycling'74 (2018). Max website. https://cycling74.com/products/max.

[Drummond, 2009] Drummond, J. (2009). Understanding interactive systems. *Organised Sound*, 14(2):124–133.

[Friedman, 1985] Friedman, M. (1985). A methodology for the discussion of contour: Its application to schoenberg's music. *Journal of Music Theory*, 29(2):223–248.

[Gauldin, 2009] Gauldin, R. (2009). *La práctica armónica en la música tonal*. Ediciones Akal, Spain.

[Guedes, 2008] Guedes, C. (2008). A lição sobre composição em tempo real. Self Published, Porto.

[Kleppinger, 2010] Kleppinger, S. (2010). Strategies for introducing pitch-class set theory in the undergraduate classroom. *Journal of Music Theory Pedagogy*, 24:131–156.

[Korg, 2013] Korg (2013). *KORG KROSS Music Workstation. Operation Guide*.

[Kostka, 2006] Kostka, S. (2006). *Materials and Techniques of Twentieth-Century Music*. Perentice Hall, New Jersey.

[Laurson and Kuuskankare, 2002] Laurson, M. and Kuuskankare, M. (2002). PWGL: A novel visual language based on common lisp, CLOS and OpenGL. In *Proceedings of the International Computer Music Conference (ICMC 2002)*, pages 142–145, Gothenburg, Sweden.

[Laurson and Kuuskankare, 2013] Laurson, M. and Kuuskankare, M. (2013). *PWGL Book*.

[López-Cano and Cristóbal, 2013] López-Cano, R. and Cristóbal, Ú. S. (2013). Investigación artísitica en música. problemas, métodos, experiencias y modelos. Conaculta, Fonca, Escola Superior de Música de Catalunya.

[Marvin, 1988] Marvin, E. (1988). *A generalized theory of musical contour: its application to melodical and rhythmic analysis of non-tonal music and its perceptual and pedagogic implications*. PhD thesis, Eastman School of Music.

[Morris, 1993] Morris, R. (1993). New directions in the theory and analysis of musical contour. *Music Theory Spectrum*, 15(2):205–228.

[Nierhaus, 2009] Nierhaus, G. (2009). *Algorithmic Composition*. Springer, New York.

[Oxford, 2018] Oxford (Consulted in may 2018). Oxford living dictionaries.

[Pachet, 2002] Pachet, F. (2002). Continuator: Musical interaction with style. In *Proceedings of the International Computer Music Conference (ICMC 2002)*, pages 211–218, Gothenburg, Sweden.

[Parncutt, 1989] Parncutt, R. (1989). *Harmony: A Psychoacoustical Approach*. Springer-Verlag, New York.

[Reich, 2004] Reich, S. (2004). Music as gradual process. In Cox, C. and Warner, D., editors, *Audio Culture*, pages 304–306. Continuum, New York.

[Rowe, 1993] Rowe, R. (1993). *Interactive Music Systems*. MIT Press, Cambridge, Massachusetts.

[Rowe, 2001] Rowe, R. (2001). *Machine Musicianship*. Massachusetts Institute of Technology, Cambridge, Massachusetts.

[Schoemberg, 1967] Schoemberg, A. (1967). *Fundamentals of Musical Composition*. Faber & Faber, London.

[Schultz, 2016] Schultz, R. (2016). Normalizing musical contour theory. *Journal of Music Theory*, 60(1):23–50.

[Straus, 2005] Straus, J. (2005). *Introduction to Post-Tonal Theory*. Perentice Hall, New York.

[Talbot et al., 2017] Talbot, P., Agon, C., and Esling, P. (2017). Interactive computer aided composition with constraints. In *Proceedings of the International Computer Music Conference (ICMC 2017)*, Shangai, China.

[Toch, 1948] Toch, E. (1948). *Elementos constitutivos de la música.* Idea Books, Barcelona.

[Winkler, 2001] Winkler, T. (2001). *Composing interactive music: techniques and ideas using Max*. Massachusetts Institute of Technology, Cambridge, Massachusetts.

[Wishart, 1996] Wishart, T. (1996). *On sonic art*. Hardwood Academic Publishers.

[Xenakis, 1992] Xenakis, I. (1992). *Formalized Music: thougth and mathematics in composition*. Pendragon Press, New York.

[Yamaha, 2013] Yamaha (2013). *Genos Digital Workstation. Owner's Manual*.